Guidewire BillingCenter®

BillingCenter System Administration Guide

RELEASE 8.0.4

Guidewire®
Deliver insurance your way™

Product Name: Guidewire BillingCenter
Product Release: 8.0.4
Document Name: *BillingCenter System Administration Guide*
Document Revision: 02-July-2015

Contents

About BillingCenter Documentation

The following table lists the documents in BillingCenter 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 BillingCenter 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 BillingCenter.
Upgrade Guide	Describes how to upgrade BillingCenter from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing BillingCenter application extensions and integrations.
New and Changed Guide	Describes new features and changes from prior BillingCenter 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 BillingCenter. The intended readers are everyone who installs the application for development or for production.
System Administration Guide	Describes how to manage a BillingCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
Configuration Guide	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files. The intended readers are all IT staff and configuration engineers.
Globalization Guide	Describes how to configure BillingCenter 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 BillingCenter.
Rules Guide	Describes business rule methodology and the rule sets in BillingCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
Contact Management Guide	Describes how to configure Guidewire InsuranceSuite applications to integrate with ContactManager and how to manage client and vendor contacts in a single system of record. The intended readers are BillingCenter implementation engineers and ContactManager administrators.
Best Practices Guide	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
Integration Guide	Describes the integration architecture, concepts, and procedures for integrating BillingCenter with external systems and extending application behavior with custom programming code. The intended readers are system architects and the integration programmers who write web services code or plugin code in Gosu or Java.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Glossary	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.

Conventions in This Document

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

Support

For assistance, visit the Guidewire Resource Portal – `http://guidewire.custhelp.com`

chapter 1

Basic Configuration Overview

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

For information about the BillingCenter installation directory contents, see “Setting Font Display Options” on page 93 in the *Configuration Guide*.

This topic includes:

The config.xml File

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

For a description of the available configuration parameters, see “Application Configuration Parameters” on page 27 in the *Configuration Guide*.

To view a read-only list of the configuration parameters and their current settings, navigate to **Server Tools** → **Info Pages** → **Configuration**. For information on the Server Tools, see “Using BillingCenter Server Tools” on page 155.

The database-config.xml File

The `database-config.xml` file stores database connection information and Data Definition Language (DDL) options. Access this file from Guidewire Studio under **configuration** → **config**. See "Configuring the Database" on page 23 in the *Installation Guide*.

Defining the Application Server Environment

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

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

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

If you intend to use clustering in your environment, read this section thoroughly before going on to read "Clustering Application Servers" on page 73. Since `isbatchserver` is only relevant in a clustered environment, a full discussion of that parameter appears in the clustering discussion.

The `gw.api.system.server.ServerUtil` Gosu class contains methods for working with system properties associated with servers. See "Reading System Properties in Plugins" on page 151 in the *Integration Guide* for information on how to use this library.

Setting Java Virtual Machine (JVM) Options

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

- `gw.bc.serverid`
- `gw.bc.env`
- `gw.bc.isbatchserver`

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

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

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

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

Specifying Environment Properties in the <registry> Element

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

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

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

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

</registry>
```

Use of the `registry` element is optional.

<systemproperty> subelement

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

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

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

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

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

<server> subelement

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

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

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

Calculating Environment Property Values

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

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

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

BillingCenter ignores any `-Dgw.bc.env` option you specify on the command prompt and sets the `env` value to `standalone`.

env Property

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

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

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

</registry>
```

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

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

isbatchserver Property

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

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

In a clustered environment, the `isbatchserver` property must resolve to `true` on at least one server. For a discussion of how the `isbatchserver` property acts in a clustered environment, see “Defining a Batch Server with the isbatchserver Environment Property” on page 80.

For more information on batch processes and work queues that distribute batch processing across multiple servers see “Batch Processing” on page 107.

serverid Property

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

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

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

</registry>
```

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

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

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

Specifying Parameters by Environment

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

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

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

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

```
-Dgw.bc.env=test
```

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

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

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

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

For a list of configuration parameters, including information about which parameters can be set by environment, see “Application Configuration Parameters” on page 27 in the *Configuration Guide*.

Providing Default Values

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

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

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

Special Environment-specific Parameters

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

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

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

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

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

Configuring an Email Server for Notifications

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

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

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

To configure email plugin parameters in Guidewire Studio

1. Start BillingCenter Studio.

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

   ```
   gwbc studio
   ```

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

See also

- “Document Management” on page 217 in the *Integration Guide*
- “Sending Emails” on page 79 in the *Rules Guide*
- “Using the Messaging Editor” on page 131 in the *Configuration Guide*
- “Using Activity Patterns with Documents and Emails” on page 378 in the *Configuration Guide*

Changing the Unrestricted User

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

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

chapter 2

Configuring Logging

This topic discusses logging in BillingCenter.

This topic includes:

Overview of BillingCenter Logging

Guidewire uses the Apache log4j utility to manage logging in BillingCenter. To determine the log4j version currently in use, check the filename of the log4j JAR file in the following location in Studio:

Project → **<BillingCenter SDK>** → **External Libraries**

The log4j file name indicates the log4j version. For example, a file name of `log4j-1.2.16.jar` indicates a log4j version of 1.2.16.

See also

- "Logging" on page 445 in the *Integration Guide*
- "Making Dynamic Logging Changes without Redeploying" on page 28

Properties File logging.properties

Properties file `logging.properties` specifies a number of system logging options for the BillingCenter application server. You access this file from the following location in Guidewire Studio:

configuration → **config** → **logging**

The `logging.properties` file uses the format specified by `log4j`. The entries in `logging.properties` control what to log and to which file to write the log. In the base configuration, BillingCenter sets the basic logging configuration to the following:

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

This setting:

- Instructs BillingCenter to send system-wide informational messages to two output points: the BillingCenter console (`Console`) and a log file (with name `DailyFileLog` in the default configuration).
- Sets the default logging level to `INFO`.

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

For detailed information about creating log4j entries, see the following Apache web site:

```
http://logging.apache.org/log4j/1.2/index.html
```

Setting the Logging Directory Path

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

Modifying the Logging Properties File

If you edit the `logging.properties` file and do not restart the server, the new logging level only take effects for new database connections. You can, however, make an immediate logging change in the BillingCenter server without having to first redeploy BillingCenter through the use of one of the following:

- Command prompt administration tool `system_tools`
- Web service `SystemToolsAPI`

See “Making Dynamic Logging Changes without Redeploying” on page 28 for more information.

Logging and the env Environment Property

If the `env` environment property is non-existent (`null`), then BillingCenter reads the logging configuration from the default `logging.properties` file. For more information on the `env` property, see “Defining the Application Server Environment” on page 14.

If the `env` environment property is non-null, however, BillingCenter tries to obtain the logging configuration from an *`env`*`-logging.properties` file in `BillingCenter/modules/configuration/config/logging` directory. For example, if you have an environment called test, BillingCenter looks for logging properties in `test-logging.properties`. If this file does not exist, then BillingCenter reads the logging configuration from the default `logging.properties` file.

See “Configuring Logging in a Multiple Instance Environment” on page 28 for more details.

Configuration Parameters Used with Logging

BillingCenter uses the following configuration parameters definitions in `config.xml` to manage aspects of the logging process:

- `LoggerCategorySource`
- `LoggersShowLog4j`

- LoggersShowPredefined

For a discussion of these configuration parameters, see “Logging Parameters” on page 51 in the *Configuration Guide*.

Specifying the Location of Log Files for the View Logs Page

BillingCenter includes a log viewer on the **Server Tools** page **View Logs**, accessible to administrators. The `guidewire.logDirectory` property in `logging.properties` specifies the location of log files for the log viewer.

Set `guidewire.logDirectory` to a directory to store the log files that you want to be visible from the **View Logs** page. Ensure that log file locations that you specify with `log4j.appender.`*`category`*`.File` are set to the same directory as `guidewire.logDirectory` for log files that you want visible from the **View Logs** page.

See “View Logs” on page 160 for more information about the `View Logs` page.

Understanding Logging Levels

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

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

Understanding Logging Categories

File `logging.properties` contains most, but not necessarily all, of the available logging categories. It is possible for both internal BillingCenter code or custom integration code to define logging categories that do not show in the logging property file. It is also possible for third-party tools to provide their own logging categories.

Along with the default logging categories, each Guidewire application has its own unique categories.

You can list the available BillingCenter Guidewire logging categories through one of the following:

- By using the `system_tools` command `-loggercats` option (from the `BillingCenter/admin/bin` directory):

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

- By using the `SystemToolsAPI` web service method `getLoggingCategories()`:

  ```
  SystemToolsAPI.getLoggingCategories()
  ```

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

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

For information about logging for plugins, see “Logging” on page 445 in the *Integration Guide*. In some cases, a BillingCenter does not have (use) a particular plugin. In those cases, the logging category exists but BillingCenter does not use the category.

The following table describes many of the major Guidewire logging categories that are available in BillingCenter.

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. Note that Guidewire ContactManager does not use this category.
`Assignment.*`	Base logging category for the assignment subsystem.
`Availability`	Logging of the determination of availability of elements in BillingCenter. BillingCenter bases the availability criteria for each element on the element type. For example, one criterion could be the effective date of an element. If the effective date is not prior to or equal to today’s date, the element would not be available.
`Configuration.*`	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 BillingCenter.
`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. BillingCenter writes logging information in this category to a separate `messaging.log` file.
`OSGi.*`	Base logging category for calls to OSGi plugins.
`PXLOGGER`	Logging for the internal Guidewire test platform.
`PerfAction.*`	Base logging category for issues related to performance.
`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`* BillingCenter writes logging information in this category to a separate `plugins.log` file. For more information on logging with plugins, see “Logging” on page 445 in the *Integration Guide*.
`Profiler`	Logging for the Guidewire Profiler. See “Guidewire Profiler” on page 175 for more information on the Guidewire Profiler.

Logging category	Description
RuleEngine	Do not use. Use RuleExecution instead.
RuleExecution.*	Base logging category for BillingCenter rule execution. Only rule execution actions generate logging events in this category. This category does not contain any information from the rules engine itself. BillingCenter writes logging information in this category to a separate ruleexecution.log file.
RuleExecutionUI	Logging of rule execution activity in the BillingCenter 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. BillingCenter 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 BillingCenter.
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 "Batch Processing" on page 107.

Setting Logging Levels by Category

The RootLogger category is the parent of all other logging categories. Setting the logging level on RootLogger causes all categories to inherit the same level, provided those categories do not set a different logging level. Set the logging level property of the different logging entries to set how much information you want sent to each type of log.

Logging levels inherit from parent categories unless the child category defines a different level. For example, setting Messaging to DEBUG also sets Messaging.Email and Messaging.Events to DEBUG, unless those categories individually define another logging level.

In the base configuration, Guidewire sets the value of rootCategory to INFO:

```
log4j.rootCategory=INFO
```

See also

- For information on the logging categories contained in logging.properties, see "Understanding Logging Categories" on page 23.
- For a complete discussion of logging levels and inheritance, refer to the Apache log4j documentation at https://logging.apache.org/.

Setting Logging Levels by System Component

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

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

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

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

You can then configure the parameters related to the `RuleEngineLog` appender to set up a log file specifically for troubleshooting rules.

Configuring Information in Log Messages

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

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

Character	Description
`%%`	Writes the percent sign to output.
`%c`	Name of the logging category. See “Understanding Logging Categories” on page 23 for categories provided with BillingCenter.
`%C`	Name of the Java class. Because the BillingCenter 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}` • ... BillingCenter uses `%d{ISO8601}` by default.
`%F`	Name of the Java source file. Because the BillingCenter logging API is a wrapper around log4j, `%F` returns a file name for the BillingCenter 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 BillingCenter logging API is a wrapper around log4j, the information returned is for the BillingCenter 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 BillingCenter logging API is a wrapper around log4j, `%L` returns a line number from the BillingCenter 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 BillingCenter 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.
`%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 For example, to include the server name, add %X{server}. For example, to include the server name, add %X{server}. There are three options for logging user information in logging patterns: user – prints the numeric opaque ID for the user userID – a unique user ID string, such as "aapplegate" userName – a real name, such as "Andy Applegate" For any of these, specify the minimum and the maximum size of the field. For example: %-16.16X{userName}. If the actual value is shorter than the minimum field size, the user identifier gets padded with spaces on the right. If the actual value is longer than the maximum size of the field, the user identifier gets truncated from the left. The user key lists a sequence number assigned to the user by the server and is not very informative. To include user login ID information, instead use the userID key.

Formatting Log Messages

You can specify the format of information in log messages by using a conversion pattern for the characters listed in “Configuring Information in Log Messages” on page 26. These conversion patterns are closely related to conversion patterns used by functions such as printf() in the C language. Add the format specification between the percent sign and the letter in the conversion pattern. The following table describes conversion patterns available with log4j.

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

Configuring Logging in a Multiple Instance Environment

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

To configure logging by using variables

1. In file `config.xml`, add an entry to the `<registry>` element for each application server that you want to log, for example:

```
<registry>
   <server env="test" serverid="testserver1"/>
   <server env="test" serverid="testserver2"/>
</registry>
```

2. In file `logging.properties`, do the following:

   a. Set the logging directory, for example:

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

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

   b. Set a value for `log4j.appender.DailyFileLog.File` that uses the `guidewire.logDirectory` and `-Dgw.bc.serverid` variables, for example:

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

3. Start each server using the system properties that set the environment and server ID values. For example, on the example development test servers, use the following commands:

```
gwbc dev-start -Dgw.bc.env=test" -Dgw.bc.serverid="testserver1"
gwbc dev-start -Dgw.bc.env=test" -Dgw.bc.serverid="testserver2"
```

As each servers starts, the server writes a log file to the common log file directory specified in `logging.properties`. Each log file includes the value of `serverid` in the log file name.

In this example, after starting the servers, the `/tmp/gwlogs/BillingCenter/logs` directory contains two log files:

- `testserver1-bclog.log`
- `testserver2-bclog.log`

Note: If you edit files `logging.properties` or `config.xml`, you must rebuild and redeploy BillingCenter for the changes to take effect. See “Deploying BillingCenter to the Application Server” on page 79 in the *Installation Guide* for more information.

Making Dynamic Logging Changes without Redeploying

You can make dynamic changes to the logging configuration without having to redeploy BillingCenter. Changing the database logging level dynamically does not make any difference to existing database connections. The new logging level only take effects for new database connections.

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

To make sure that the logging level is set consistently for all database connections, set the log level in `logging.properties` and restart the server.

Reloading the Logging Configuration

It is possible to make an immediate logging change in the BillingCenter server without having to first redeploy BillingCenter. First, update the `logging.properties` file with your changes. Then, update the logging configuration on the server by using one of the following:

- Command prompt administration tool `system_tools`
- Web service `SystemToolsAPI`

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

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

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

```
SystemToolsAPI.reloadLoggingConfig()
```

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

See also

- "System Tools Command" on page 191
- "System Tools Web Services" on page 129 in the *Integration Guide*

Temporarily Changing a Logging Level

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

- By using the `system_tools` command prompt utility
- By using the `SystemToolsAPI` web service
- By setting the log level on the **Set Log Level** info page

You must supply a `String` representation of either the logging category for category-based logging or the Java class name for class-based logging. For the root logger, specify `RootLogger` for the logger name. The change in the logging level persists only while the BillingCenter server is running.

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

To set a logging level using the `system_tools` command, use the following command from the `admin/bin` directory:

```
system_tools -updatelogginglevel logger level
```

To set a logging level using the `SystemToolsAPI` web service, call the following method on the `SystemToolsApi` web service:

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

To set a logging level directly within BillingCenter, use the **Set Log Level** page in BillingCenter that is available to administrators. See "Set Log Level" on page 160 for details.

chapter 3

Configuring and Maintaining the BillingCenter Database

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

This topic includes:

See also

- "Configuring the Database" on page 23 in the *Installation Guide*
- "Configuring a Database Connection" on page 62 in the *Installation Guide*
- See the *Guidewire Platform Support Matrix* for current system and patch level requirements. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at `https://guidewire.custhelp.com/app/resources/products/platform`.

Database Best Practices

Guidewire recommends the following best practices for the database:

- If you need to perform any database maintenance tasks, such as applying a patch, shut down application servers running BillingCenter that are attached to the database. Restart the application servers after the database maintenance is complete.
- For Oracle databases, keep the Oracle default settings as much as possible. For example, do not set a 4K blocksize. Consult with Guidewire if you want to change the default Oracle settings.
- Do not insert directly into tables managed by BillingCenter. This can cause the data distribution tool to fail and cause other problems. See “Guidewire Database Direct Update Policy” on page 32.
- Do not add lots of `mediumtext` and `CLOB` columns to a table.
- Do not add an index outside of BillingCenter and not declare it in an extension file.
- Monitor how storage performs. If I/O (input/output) times are slower than 10ms, something is wrong.
- Monitor tablespace size allocations and disk space, so BillingCenter does not run out of space.
- Back up the database periodically to support disaster recovery options. See “Backing up the BillingCenter Database” on page 34.
- Update database statistics periodically so that the query optimizer selects an efficient plan for executing application queries. See “Configuring Database Statistics” on page 39.
- Run consistency checks on the database, especially after importing data. See “Checking Database Consistency” on page 36.

Guidewire Database Direct Update Policy

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

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

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

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

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

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

Configuring Connection Pool Parameters

If you experience slow performance, it could be that the BillingCenter 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, BillingCenter periodically tests connections in the connection pool and evicts idle connections and those that fail with an exception when tested with a simple query. You can configure this behavior and set other connection pool parameters by modifying or adding attributes to the `<dbcp-connection-pool>` element within the `<database>` element of `database-config.xml`. Access this file from the Guidewire Studio **Project** window under **configuration** → **config**.

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

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

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

The parameters listed previously apply if you are using the default connection pool. If you instead use the application server connection pool, these settings do not apply. Configure the application server connection pool through the administration console provided with the application server. See “Configuring BillingCenter to Use a JNDI Data Source” on page 68 in the *Installation Guide*.

Backing up the BillingCenter Database

BillingCenter 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 BillingCenter database. You can take a hot backup while users are still accessing BillingCenter. Read your database documentation to understand the risks involved in hot backups. If you plan on taking a cold backup, you must put the application server in maintenance mode before performing the backup.

After restoring a BillingCenter database from a backup, instruct BillingCenter to rebuild its database statistics. See “Configuring Database Statistics” on page 39.

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

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

Understanding and Authorizing Data Model Updates

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

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

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

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

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

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

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

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

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

The database update takes a number of actions that can impact database statistics. The upgrade might recreate tables, drop indexes and create new indexes. The update process writes an `updatedatabasestatistics.sql` file to the `work` directory on the application server. The contents of the file are dependent on the SQL executed during the update and the statistics parameters of the `<database>` element. After an update, run the `updatedatabasestatistics.sql` script to update database statistics. See “Configuring Database Statistics” on page 39 for more information about setting the statistics parameters.

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

For more details on configuration changes that require database updates, see “Modifying the Base Data Model” on page 205 in the *Configuration Guide*.

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

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

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

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

To download upgrade instrumentation details

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

Checking Database Consistency

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

Guidewire Recommendations

Guidewire recommends that you run the database consistency checks on a regular basis to identify potential problems. The following is a simple schedule:

- Before importing data into a production database
- Before and after a database upgrade
- Weekly after a new BillingCenter deployment
- Monthly after stabilization of a new BillingCenter deployment
- Monthly while testing imported data that you are converting from a legacy system
- Monthly after you have moved the converted data to a production database

Performance Issues

For very large databases, running consistency checks can have a major negative effect on performance. Some consistency check queries use a large amount of temporary space on Oracle. BillingCenter includes two parameters for very large databases that allow some of these queries to be split into multiple smaller queries. Each of these smaller queries check only a portion of the database. Only set these parameters if you have a very large database. The configuration parameters are:

- `MaxTAccountsInConsistencyCheck`
- `MaxTransactionsInConsistencyCheck`

Both parameters have a default value of 0, which configures BillingCenter to not split queries.

Running Consistency Checks from BillingCenter

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

See "Consistency Checks" on page 161 for more information.

Running Consistency Checks with System Tools

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

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

The `system_tools` utility is located in the `BillingCenter/admin/bin` folder. See "System Tools Command" on page 191 for more information.

This tool has two optional arguments:

```
-checkdbconsistency tableSelection checkTypeSelection
```

The *`tableSelection`* option uses the following arguments:

- `all` – Run consistency checks on all tables.
- *`table name`* – The name of a single table on which to run checks.
- `tg.`*`table group name`* – The name of a table group. Table groups are defined in the `<database>` element of `database-config.xml`. For more information, see "Defining Table Groups" on page 65 in the *Installation Guide*.
- `@`*`file name`* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with `tg.`, such as `tg.MyTableGroup`. You can combine table groups and individual table names in the same file.

The *`checkTypeSelection`* option uses the following arguments:

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

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

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

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

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

Results of the consistency checks are available from the **Consistency Checks** page. See "Consistency Checks" on page 161.

The BillingCenter log lists a check type for each consistency check that runs. Each check type in the log can have a value of either `0` or `1`. A value of `0` indicates that BillingCenter expects the check to return zero results if the database is consistent. A value of `1` indicates that BillingCenter expects the check to have a different return value. BillingCenter uses the check type for custom checks. If a check type `0` fails, the log records a failure description that BillingCenter 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 BillingCenter log is not the same as the check type shown in the Typelists section of the *BillingCenter Data Dictionary*.

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

- Run consistency checks on fewer tables at a time by using the arguments shown previously.
- Increase the number of worker instance threads used by the consistency check work queue. See "Configuring Number of Threads for Consistency Checks" on page 38.

Running Consistency Checks as the Server Starts

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

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

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

Configuring Number of Threads for Consistency Checks

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

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

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

In this example, the consistency check work queue is configured to use five worker threads and for each worker to check out ten work items at a time. Consistency checks run only the batch server. If you include the optional `<server>` attribute or configure the work queue with multiple servers, BillingCenter ignores them.

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

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

See also

- "Configuring Work Queues" on page 116

Configuring Database Statistics

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

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

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

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

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

For Oracle, if you would like to run the incremental database statistics process after running the `table_import -integritycheckandload` command, ensure that you flush the monitoring information using the procedure `DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO`.

For SQL Server, Guidewire requires that you set `Auto Create Statistics` and `Auto Update Statistics` to `true` on the database account used for BillingCenter.

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 `table_import -integritycheckandload` or `zone_import` command, or if there are performance problems. Under normal operating conditions, you do not need to update database statistics on the BillingCenter server often. If you encounter performance problems or degradation related to the database, check the **Database Statistics** page on the **Info Pages** section of the **Server Tools**. If the page shows suspicious or inaccurate statistics, update database statistics. If the data change is high, consider using a weekly or biweekly schedule for updating statistics.

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

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

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

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

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

As you import data into tables from an external source by using the `table_import -integritycheckandload` command, BillingCenter validates the staging tables against the data model and the production tables. To avoid the formation of bad queries prior to loading new data, BillingCenter updates statistics for the involved tables before and after the load. For optimum performance, generate full database statistics after running the `integritycheckandload` process. For more information on the `table_import` tool, see "Table Import Command" on page 195.

Using the **Database Statistics** page, you can see if any statistics are out of date. See "Database Statistics" on page 167. Guidewire recommends that you archive database statistics as standard practice. This ensures that you have a record of the database history that can be reviewed if necessary.

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

Commands for Updating Database Statistics

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

Statistics type	Command

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

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

Staging Generates database statistics for staging tables.	– To generate statistics for staging tables: `table_import -password password -updatedatabasestatistics`

Configuring Database Statistics Generation

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

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

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

```
      name="string"
      numbuckets="integer"
      databasedegree="integer"
      samplingpercentage="integer"/>
     ...
  </tablestatistics>
  ...
</databasestatistics>
```

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

```
<databasestatistics samplingpercentage="0" databasedegree="4"
 incrementalupdatethresholdpercent="15">
  <tablestatistics name="bc_table1" samplingpercentage="100" databasedegree="4">
    <histogramstatistics name="scheduledsenddate" numbuckets="254"/>
    <indexstatistics samplingpercentage="50">
      <keycolumn name="publicid" keyposition="1"/>
      <keycolumn name="retired" keyposition="2"/>
    </indexstatistics>
  </tablestatistics>
  <tablestatistics name="bc_table2" action="delete" />
</databasestatistics>
```

The above example configures the following database statistic generation behavior:

- Collect statistics on all BillingCenter tables using the automatic sampling size and with a degree of parallelism of `4`.
- The `dbms_stats` command for table `bc_table1` samples 100% and uses a parallel degree of `4`.
- The configuration defines a histogram with 254 buckets on `cc_check.scheduledsenddate`.
- The index on `bc_table1` (`publicID, retired`) is sampled at 50% and statistics for the index are gathered.
- BillingCenter deletes statistics on `bc_table2` due to the attribute `action="delete"`.

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

<databasestatistics>

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

`databasedegree`	On Oracle, this attribute controls the degree of parallelism for each individual statement. The default is `1`. BillingCenter uses the value of this attribute for all statements. SQL Server ignores the `databasedegree` attribute.
`samplingpercentage`	On Oracle, this attribute controls the value of the `estimate_percent` parameter in the `dbms_stats.gather_table_stats()` SQL statements. You can set `samplingpercentage` to either an integer from `1` to `100` to directly set the `estimate_percent` value, or set `samplingpercentage` to `0` to set `estimate_percent` to `AUTO_SAMPLE_SIZE`. The default value is `0`. BillingCenter uses the database default for `dbms_stats.gather_index_stats()` statements. On SQL Server, the `samplingpercentage` attributes controls the value of the `WITH FULLSCAN/SAMPLE PERCENT` clause in the `UPDATE STATISTICS` statements. A value of `100`, the default, translates into `WITH FULLSCAN`, as does a value of `0`.

incrementalupdatethresholdpercent	Specifies the percentage of table data that must have changed since the last statistics process for the incremental statistics generation batch process to update statistics for the table.
numappserverthreads	On both Oracle and SQL Server, the numappserverthreads attribute controls the number of threads that are used to update database statistics for staging tables during import only. This import is launched using the table_import command. See “Table Import Command” on page 195. The value defaults to 1. If the value is greater than 1, then the application server assigns a table at a time to each thread as the thread becomes available. Each thread executes all of the database statistics statements for its assigned table. For all other statistics generation operations, set the number of threads by specifying the number of workers for the database statistics work queue. Set the instances attribute on the workers subelement of the work-queue element for the database statistics work queue. This element has workQueueClass="com.guidewire.pl.system.database.dbstatistics.DBStatisticsWorkItemWorkQueue".

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

<tablestatistics>

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

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

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

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

The default value is update.

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

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

<indexstatistics>

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

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

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

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

<histogramstatistics>

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

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

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

The <histogramstatistics> element has the following format:

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

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

Notes

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

Configuring Number of Threads for Statistics Generation

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

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

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

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

For more information about work queue configuration, see “Scheduling Work Queue Writers and Batch Processes” on page 113.

Checking the Database Statistics Updating Process

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

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

Canceling the Database Statistics Updating Process

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

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

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

Purging Old Workflows and Workflow Logs

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

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

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

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

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

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

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

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

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

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

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

Resizing Columns

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

To resize columns

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

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

IMPORTANT Guidewire does not support resizing base columns.

chapter 4

Data Change API

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

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

This topic includes:

Data Change API Overview

In typical conditions, BillingCenter 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.

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

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

Separation of Roles

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

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

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

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

Preserving Results

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

Replay Prevention

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

Typical Use of the Data Change API

There are several steps in using the data change API:

- “Write Data Change Code” on page 48
- “Register a Data Change” on page 49
- “Run Data Change Code” on page 50

Write Data Change Code

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

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

To persist changes to the database, use the `gw.Transaction.Transaction` class and its method `runWithNewBundle`. See “Running Code in an Entirely New Bundle” on page 351 in the *Gosu Reference Guide*. You pass the method a block that runs code. If the block does not throw an exception, BillingCenter persists any data changes from your Gosu block to the database. If the block throws an exception, no changes persist to the database.

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

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

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

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

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

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

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

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

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

})
```

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

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

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

Register a Data Change

There are two ways to register your data change code

- The command prompt tool `BillingCenter/admin/bin/data_change.bat`
- The WS-I web service `DataChangeAPI`

The data change registration details vary between these two variants.

In all cases, before proceeding you must have:

- Data change code in the form of a Gosu script that you have already tested in your development environment. See “Write Data Change Code” on page 48.
- A human-readable description for your data change
- A unique reference ID that you create to represent this data change

Register a Data Change From Command Prompt

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

To register a data change from command prompt

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

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

For example:

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

For complete documentation on all command prompt options, see “Data Change Command Prompt Reference(data_change.bat)” on page 51.

The script outputs results such as:

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

Register a Data Change From a Web Service

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

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

- “Register a Data Change From Command Prompt” on page 50
- “Data Change Command Prompt Reference(data_change.bat)” on page 51

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

For example:

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

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

Run Data Change Code

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

To run a data change

1. Confirm that someone created and registered a data change as described in "Write Data Change Code" on page 48. You must know the reference ID for the registered data change.
2. Log into BillingCenter as an administrator. Note that the user that created and registered the data change may not be the same person as the person running the data change in the production environment. You can define these roles to have different permissions.

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

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

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

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

   ```
   REFID1235 finished okay
   ```

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

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

   If you need to re-run a successful data change, you must first re-register the script with a new reference ID. This is a requirement preserves the integrity of the results log. See "Register a Data Change" on page 49.
9. If the data change appears safe in your development environment, carefully register and run the data change on the production server.

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

Data Change Command Prompt Reference(data_change.bat)

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

See also

- "Data Change API Overview" on page 47
- "Typical Use of the Data Change API" on page 48

- “Data Change Web Service Reference (DataChangeAPI)” on page 52

Option for data_change.bat	Description
Data change actions (choose one)	
`-edit` *`refID`*	This option indicates you want to create a new data change or edit an existing data change. Include 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: • include the `gosu` argument to include your Gosu data change code • include the `description` argument for a description
`-discard` *`refID`*	This option indicates you want to discard a data change that you already registered. Pass a data change reference ID (*`refID`*). You cannot discard a data change that was already run.
`-result` *`refID`*	This option indicates you want the result of a data change that you already registered. Pass a data change reference ID (*`refID`*). If a user attempted to run it and there were parse errors, the results include the errors.
`-status` *`refID`*	This option indicates you want the status of a data change that you already registered. Pass a data change reference ID (*`refID`*). The tool prints the status, which is `Open`, `Discarded`, `Executing`, `Failed`, or `Completed`.
Data	
`-description` *`description`*	A human-readable description (*`description`*) of the change. Include this option when you use the `edit` argument. For testing, the description is optional. For production use, include the description. Put quotes around the description to permit space characters in the description.
`-gosu` *`filepath`*	The full path name (*`filepath`*) to a Gosu script. Include this option when you use the `edit` argument. You can use a full path name, or a relative path that is relative to the current working directory.
Server authentication (required)	
`-server` *`url`*	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-user` *`user`*	The user (*`user`*) to use to run this process. The user must have permission `wsdatachangeedit`.
`-password` *`password`*	Specifies the *`password`* to use to connect to the server. BillingCenter requires the password.

Data Change Web Service Reference (DataChangeAPI)

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

See also

- “Data Change API Overview” on page 47
- “Typical Use of the Data Change API” on page 48a

- “Data Change Command Prompt Reference(data_change.bat)” on page 51

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

chapter 5

Managing BillingCenter Servers

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

This topic includes:

Stopping the BillingCenter Application

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

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

To stop work queues from the Batch Process Info page

1. Press `ALT+SHIFT+T` to display the **Server Tools** tab.
2. Click **Batch Process Info** if not already on the **Batch Process Info** page.

3. For any process that has a **Status** of **Running**, click the **Stop** button in the **Action** column. Wait for all processes to have a **Status** of **Inactive** before stopping BillingCenter.
4. For any process that has a **Next Scheduled Run** time that is before the time that you will stop BillingCenter, click **Stop** in the **Schedule** column. This disables the schedule for the current BillingCenter session. The schedule is enabled again when you restart BillingCenter, according to the settings in `scheduler-config.xml`. See "Scheduling Work Queue Writers and Batch Processes" on page 113.

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

Server Modes and Run Levels

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

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

Important Server Mode Caveats

The following caveats are important to remember 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 Test Mode

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

- You can adjust the testing system clock by using the `setCurrentTime` method on the `ITestingClock` plugin. See "System Clock" on page 184. Also see "Testing Clock Plugin (Only For Non-Production Servers)" on page 266 in the *Integration Guide*.
- Both the **Server Tools** and **Internal Tools** tabs are available. In production mode, only the **Server Tools** tab is available. For information about these tools. See "Using BillingCenter Server Tools" on page 155 and "Using BillingCenter Internal Tools" on page 183.
- BillingCenter prints a message to the console during startup indicating that the server is running in test mode.
- The browser title bar for a browser connected to BillingCenter indicates that BillingCenter is in test mode.

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

Server Modes as a Safety Feature

Guidewire provides these modes as a safety precaution so that development tools are not used on a production server. Some system functions are useful for development, but are not appropriate, or even dangerous, if used in a production environment. In development and test mode, both the **Server Tools** and **Internal Tools** tabs are available. In production mode, only the **Server Tools** tab is available. See "Using BillingCenter Server Tools" on page 155 and "Using BillingCenter Internal Tools" on page 183.

An example is the `ITestingClock` plugin that provides functionality for setting the current time and is critical for testing time-sensitive processes. You can also use this plugin to modify the current time in a running server for demonstrations. However, use of this plugin in a production environment could have disastrous results. Therefore, you can only use this plugin when the server is in development or test mode. Test mode is identical to

production mode except that in test mode you can adjust the testing system clock. See “Testing Clock Plugin (Only For Non-Production Servers)” on page 266 in the *Integration Guide*. See also “System Clock” on page 184.

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

Server Run Level Implications

The BillingCenter server can also be put into `MULTIUSER`, `DAEMONS`, and `MAINTENANCE` run levels. See “Using the Maintenance Run Level” on page 59. These run levels are independent of the mode. The combination of mode and run level determines the availability of functionality, such as the user interface and web services. For details, see “Server Modes and Run Levels” on page 56.

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

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

Setting the Server Mode

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

Control the mode through the system parameter:

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

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

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

or

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

or

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

BillingCenter ignores this parameter on the QuickStart server.

Determining Server Mode

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

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

Setting the Server Run Level

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

To set the server run level to multiuser

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

To set the server run level to daemons

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

To set the server run level to maintenance

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

You can also set the server run level using the `SystemToolsAPI` web service. You cannot set the server to the `SHUTDOWN`, `GUIDEWIRE_STARTUP` or `NONE` run level. However, `SystemToolsAPI.getRunlevel()` can report these run levels. See “Getting and Setting the Run Level” on page 129 in the *Integration Guide*.

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

Determining the Server Run Level

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

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

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

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

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

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

Using the Maintenance Run Level

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

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

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

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

Monitoring the Servers

You can use an HTTP ping utility provided by Guidewire to check server status. See “Checking Node Health” on page 86.

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 BillingCenter does not run out of disk space for logs. Archive and truncate system logs periodically to prevent the BillingCenter 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 (534841524544325 54E54494D450E43505001A8)
```

See “Configuring the Application Server” on page 15 in the *Installation Guide* and documentation provided with the application server for instructions on increasing the maximum heap size.

Monitoring with WebSphere

You can monitor the server’s status from the WebSphere console. To view the server’s status, select the BillingCenter node 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 backup the server before performing any major system maintenance.

Monitoring and Managing Event Messages

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

Monitor message traffic to ensure that the integration is running smoothly. This section discusses topics in monitoring and managing event messages. For more information about messages, including how to create message destination objects, see “Messaging and Events” on page 303 in the *Integration Guide*.

How BillingCenter Processes Messages

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

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

For example:

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

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

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

Unsent	The message has not been sent. The message might be waiting on a prior message. Or, the destination might not be processing messages because it is suspended. Or, the destination is falling behind. BillingCenter can generate messages very quickly
Needing Retry	Waiting to attempt a retry. BillingCenter attempted to send the message but the destination threw an exception. If the exception was retryable, BillingCenter automatically attempts to retry the send before turning the message into a failure. BillingCenter 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	BillingCenter is waiting for an acknowledgement.
Messages Failed	A message can fail for several reasons. • A processing error, the destination did not process the message successfully. • The destination returns a negative acknowledge (nack) indicating that the message failed. • The message was embedded in a series of messages, one or more of which failed.

If BillingCenter 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 BillingCenter receives a processing error that is not retryable, BillingCenter also suspends the destination and waits for you to clear the error. To clear an error, either manually retry the send or skip the message. Do this from the user interface or from the command prompt.

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

Working with the Destinations Page

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

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

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

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

Configuring Message Destinations

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

Tuning Message Handling

A BillingCenter 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 BillingCenter places the messages in the queue.

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

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

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

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

System Users

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

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

BillingCenter also requires `sys`, the system user. This is the user BillingCenter uses to do automated work such as running batch processing, messaging polling, and server startup. Each time BillingCenter 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.

IMPORTANT Do not rename or delete the `sys` user.

Configuring Minimum and Maximum Password Length

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

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

Configuring Client Session Timeout

BillingCenter creates a session for each browser connection. BillingCenter uses the application server's session management capability to manage the session. Each session receives a security token that BillingCenter 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` property in `config.xml`. This value sets the session expiration timeout globally for all BillingCenter browser sessions.

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

Level	Description
Server	The session timeout to use for all applications on the server if a timeout value is not specified at a higher level.
Enterprise application	The session timeout specified at the enterprise application level. You can specify this value at the EAR file level. You can set the enterprise application session timeout value to override the server session timeout value.
Web application	The session timeout specified at the web application level. You can specify this value at the WAR file level.You can set the web application session timeout value to override the enterprise application and server session timeout values.
Application level	The session timeout specified in the application `web.xml` file. BillingCenter does not specify a session timeout in `web.xml`.
Application code	An application can override any other session timeout value. BillingCenter uses the session timeout value specified by the `SessionTimeoutSecs` parameter in `config.xml`.
User	An administrator can set the session expiration timeout value on an individual user basis, using the **Session timeout** field on the BillingCenter **User** page.

See also

- "SessionTimeoutSecs" on page 59 in the *Configuration Guide*

Avoiding Session Replication

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

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

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

Application Server Caching

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

This topic includes:

Cache Management

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

- A stale timeout mechanism ensures that the server does not use excessively old object entries. An object is stale if it has not been refreshed from the database within a configurable amount of time. Upon accessing a cache entry, the server calculates the duration since the object was last read from the database. If that duration exceeds the stale time, the server refreshes the cache entry from the database. To avoid increased complexity, BillingCenter prefers this mechanism over evicting objects upon stale timeout. You can set a default stale time by adjusting the `GlobalCacheStaleTimeMinutes` parameter in `config.xml`.
- An evict timeout mechanism removes old objects from the cache. For example, a bean has an evict time of 15 minutes and a stale time of 30 minutes. If the server uses the object once every 14 minutes, BillingCenter 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 maximum value for this parameter is the smaller of `15` and `GlobalCacheStaleTimeMinutes`.
- Upon reception of an inter-cluster message indicating that an object value was changed in another node, the server marks the corresponding entry in the cache obsolete and available for reuse.

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

Caching and Stickiness

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

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

Concurrent Data Change Prevention

Different users, either on the same application server instance or on different ones, might change objects concurrently. Guidewire implements a data versioning mechanism to prevent corruption in such cases. As BillingCenter updates an object value to the database, BillingCenter compares a counter associated with the object to the counter in the database. A counter value mismatch indicates that the object was concurrently changed. In such case, BillingCenter rejects the change and the cache mechanism throws a concurrent change exception. BillingCenter presents the user who initiated the concurrent change with the error and reloads the latest data. The user can then reapply the changes. Furthermore, BillingCenter commits changes in an atomic bundle, ensuring transactional integrity. Therefore, BillingCenter enforces protection against concurrent data changes across the whole transaction. This mechanism is a standard design pattern called optimistic locking.

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

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

Caching and Clustering

For information on how Guidewire clusters handle caching, see “Cache Usage in Guidewire Clusters” on page 77.

Performance Impact

This section distinguishes two caches:

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

Proper caching behavior is critical to performance. “Analyzing and Tuning the Application Server Cache” on page 65 describes how to size a cache correctly.

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

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

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

Cache Thrashing

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

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

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

See "Detecting Cache Thrashing" on page 67.

Cache Impact on Memory Utilization

The maximum size of the cache is dependent on cache parameters. See "Analyzing and Tuning the Application Server Cache" on page 65. The application server cache grows in size to reach a maximum specified by cache sizing parameters. Java does not provide a good means to estimate the memory usage of objects. Therefore, the maximum size of a cache cannot be reliability estimated. If the cache size exceeds the maximum heap size, the application eventually runs out of memory.

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

Set the cache as large as needed, but no larger. Monitor garbage collection to extrapolate memory usage patterns and garbage collection statistics. See "Analyzing Server Memory Management" on page 68.

Analyzing and Tuning the Application Server Cache

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

Parameter	Description
`ExchangeRatesRefreshIntervalSecs`	The number of seconds between refreshes of the exchange rates cache. This is a specialized cache only for exchange rates. See "Special Caches for Rarely Changing Objects" on page 67.
`GlobalCacheActiveTimeMinutes`	Time, in minutes, that BillingCenter considers cached objects active. The cache gives higher priority to preserving these objects. This can be thought of as the period that items are being heavily used, for example, how long a user stays on a page. Set `GlobalCacheActiveTimeMinutes` to a value less than `GlobalCacheReapingTimeMinutes`.
`GlobalCacheDetailedStats`	Boolean that specifies whether to collect detailed statistics for the global cache. Detailed statistics are data that BillingCenter collects to explain why items are evicted from the cache. Basic statistics, such as miss ratio, are still collected regardless of the value of `GlobalCacheDetailedStats`. Disabling collection of detailed cache statistics can sometimes improve performance. `GlobalCacheDetailedStats` is set to `false` by default. Set the parameter to `true` to help tune your cache. If the `GlobalCacheDetailedStats` parameter is set to `false`, the **Cache Info** page does not include the **Evict Information** and **Type of Cache Misses** graphs. At runtime, use the **Management Beans** page to enable the collection of detailed statistics for the global cache. See also: • "Cache Info" on page 173 • "Management Beans" on page 171

Parameter	Description
GlobalCacheReapingTimeMinutes	Time, in minutes, since last use of a cached object before BillingCenter considers the object eligible for reaping. This can be thought of as the period during which BillingCenter 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, BillingCenter never evicts the cache entry, but the entry does eventually become stale. GlobalCacheReapingTimeMinutes is initially set to 15 minutes. The minimum value for this parameter is 1 minute. Since the eviction thread only runs once per minute, a smaller value would not make sense. The maximum value for this parameter is 15 minutes.
GroupCacheRefreshIntervalSecs	The number of seconds between refreshes of the groups cache. This is a specialized cache only for groups. See “Special Caches for Rarely Changing Objects” on page 67.
GlobalCacheSizeMegabytes	Maximum amount of heap space used to store cached entities, expressed as a number of megabytes. This parameter supersedes the value of GlobalCacheSizePercent. At runtime, you can use the **Cache Info** or **Management Beans** page to modify this value. See also: • “Cache Info” on page 173 • “Management Beans” on page 171
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 BillingCenter 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, BillingCenter prefers this mechanism over evicting objects upon stale timeout. At runtime, you can use the **Cache Info** or **Management Beans** page to modify this value. See also: • “Cache Info” on page 173 • “Management Beans” on page 171
GlobalCacheStatsWindowMinutes	This parameter denotes a period of time, in minutes, that BillingCenter uses for two purposes: • how long to preserve the reason that BillingCenter evicted an object, after the event occurred. When a cache miss occurs, BillingCenter reports the reason on the **Cache Info** page. • the period for which to display statistics on the chart on the **Cache Info** page. For more information on the **Cache Info** page, see “Cache Info” on page 173.

Parameter	Description
ScriptParametersRefreshIntervalSecs	The number of seconds between refreshes of the script parameters cache. This is a specialized cache only for script parameters. See “Special Caches for Rarely Changing Objects” on page 67.
ZoneCacheRefreshIntervalSecs	The number of seconds between refreshes of the zones cache. This is a specialized cache only for zones. See “Special Caches for Rarely Changing Objects” on page 67.

Analyzing Cache Settings

See “Cache Info” on page 173 for information on how to view cache performance. The cache performance information includes the number of objects currently in the cache, the number of objects evicted from the cache, and more.

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

Detecting Cache Thrashing

You can find evidence of cache thrashing by:

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

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

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

Special Caches for Rarely Changing Objects

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

- Exchange rates
- Groups
- Script parameters
- Zones

These caches periodically refresh the entire set of the rarely changing object. This prevents the application server from querying the database each time BillingCenter accesses one of these objects, thereby improving performance. For each of these special caches, you can set the refresh interval. See “Analyzing and Tuning the Application Server Cache” on page 65.

Analyzing Server Memory Management

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

This topic describes Java platform garbage collection analysis.

This topic includes:

Memory Usage Logging

The memory usage logging message looks like the following:

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

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

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

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

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

This logging line cannot provide more detailed information, such as "used active" versus "used stale", without actually running the garbage collector. To do so just for the sake of more detailed logging would interfere with the optimal pattern of garbage collection and is not supported. Turn on garbage collector logging to get more detailed information on the memory usage of the application, as it is currently configured. See "Enabling Garbage Collection" on page 68.

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

Enabling Garbage Collection

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

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

IBM JVM

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

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

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

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

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

Oracle Java Hotspot VM

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

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

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

These options provide the following:

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

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

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

Analyzing Garbage Collector Behavior

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

IBM Support Assistant

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

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

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

HPjmeter and GCViewer

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

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

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

Refer to the following URLs for more information:

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

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

Analyzing a Possible Memory Leak

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

Garbage collection logs might show that memory usage grows significantly over time, resulting in a lack of available memory. This condition is commonly described as a memory leak. 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 BillingCenter 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.

Investigation of memory leaks differs slightly per JVM platform.

Common approach

Various options exist to generate a heap dump:

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

  These options are combined with options instructing the JVM to generate the heap dump at a specific directory location.
- Tools can connect to a running JVM. Such tools provide the option to trigger a heap dump.

Generating Heap Dumps with IBM JDK

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

Generating Heap Dumps with Oracle HotSpot JDK

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

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

Additional Recommendations

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

- Heap dump generation frequently fails because the single file generated is very large and the supporting environment is configured to prevent regular accounts to create such large files. Therefore, some configuration is generally required to allow the account running the node to create such large files.
- The generation of a heap dump during out-of-memory conditions is sometimes challenging. As a JVM is reaching maximum memory utilization, it generally experiences severely degraded performance. As the pace of the leak decreases gradually, the occurrence of the out-of-memory condition might take an inordinate amount of time. This length of time might be incompatible with the need to restore performance for users or processes.
- Windows only: Windows does not support signals. Therefore, generating a heap by starting the JVM with a heap dump on `CTRL-BREAK`, depends on the capacity to send a `CTRL-BREAK`. You cannot send a `CTRL-BREAK` to a JVM started as a background process. Therefore, for the time of the investigation, start the JVM from a command 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 “Application Server Caching” on page 63.

Heap Dump Generation and Analysis Tools

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

IBM-only Tools

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

Oracle-only Tools

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

Generic tools

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

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

Profiling

Java profilers are available for two main purposes:

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

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

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

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

Tracking Large Objects

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

chapter 6

Clustering Application Servers

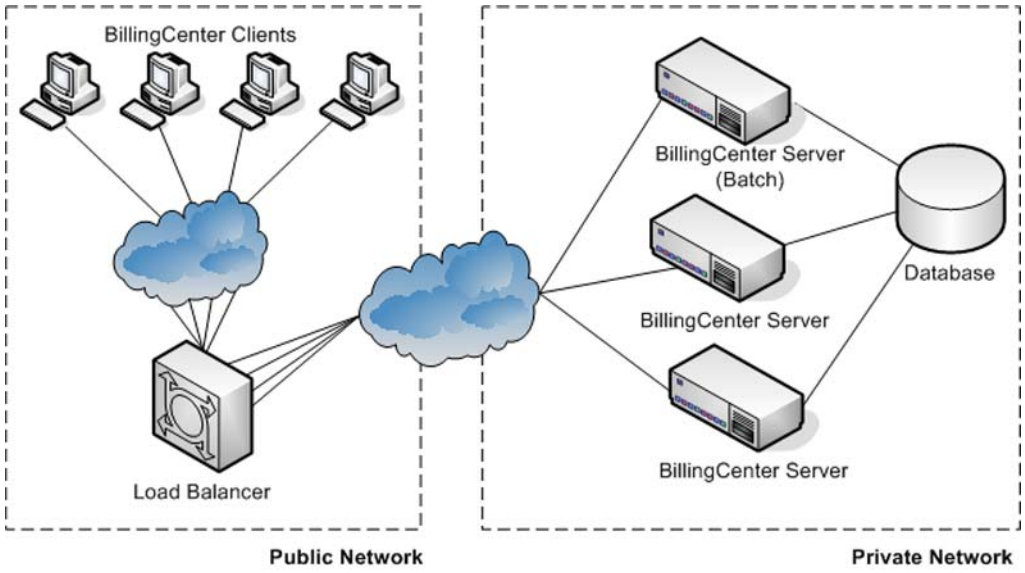
To improve performance and reliability, you can install multiple BillingCenter servers in a configuration known as a cluster. A cluster distributes client connections among multiple BillingCenter servers, reducing the load on any one server. If one server fails, the other servers seamlessly handle its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

Also see “Considerations for a Clustered Application Server Environment” on page 16 in the *Installation Guide*.

This topic includes:

Overview of Clustering

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



Planning a BillingCenter Cluster

Plan the cluster so that if any one server fails, the other servers in the cluster can handle its traffic without being overwhelmed. BillingCenter 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 BillingCenter 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 BillingCenter server. The load balancer must implement *session affinity*, meaning that it must route connections from the same user session to the same BillingCenter 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

- "Load Balancers" on page 19 in the *Installation Guide*
- "Considerations for a Clustered Application Server Environment" on page 16 in the *Installation Guide*

The Cluster Batch Server

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

In general, start the batch server first. If it fails, either restart the batch server, start another batch server or use the Management Beans section of Server Tools to designate another node as the batch server. See “Management Beans” on page 171. If another server goes down that is not the batch server, you can restart that server without restarting each computer in the cluster.

Special Considerations Regarding BillingCenter Batch Servers

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

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

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

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

Notes:

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

The Batch Server and Script Parameters

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

JGroups Clustering

Guidewire bases its clustering on JGroups, which is a toolkit for reliable multicast communication.

In JGroups clustering, there is the concept of a cluster coordinator. The coordinator is responsible for cluster formation and the merging of a split cluster. The first node to start in the cluster becomes the coordinator. If that node fails, some communication will happen to identify another coordinator. In certain rare cases, for example, with network issues, it is possible for several nodes to become a coordinator. After the issue is resolved, the various coordinators will identify the presence of multiple coordinators and a single node becomes the coordinator.

You must start the batch server first if you deploy a new Guidewire release, or, if you deploy a custom configuration with data model changes. This is because the batch node is the only node that can apply data model changes. Aside from this case, it is possible to start Guidewire application servers in any order. However, most installations typically start the batch server first in all cases. As a result, the batch server is frequently the cluster coordinator, even though there is no requirement to have the cluster coordinator be the batch server node.

With JGroups clustering, each node maintains a view of the other nodes in the cluster. Each node regularly sends a message validating its presence. If a node fails to send its validation message for a certain number of retries, the other nodes will consider that the non-responsive node has left the cluster.

Each node saves each sent validation message in local memory (in a retransmission table) until all other nodes in the cluster acknowledge receipt of the sent message. This means that for a multicast message, all cluster nodes need to acknowledge the receipt of the message. If a node does not acknowledge the message, and, instead, leaves the cluster, the remaining cluster nodes delete the sent messages from the missing node from their retransmission tables. The safekeeping of these sent validation messages increases memory usage.

See also

- “Using ENCRYPT JGroups Protocol” on page 16 in the *Installation Guide*
- “Understanding Logging Categories” on page 23

Cluster Communication

Guidewire clusters use both unicast and multicast communication modes:

- In *unicast* mode, communication occurs between a client and a server in an inherently point-to-point fashion. Thus, network communication uses unicast mode only if two applications need to communicate with each other, such as an application server with the database.
- In *multicast* mode, applications instances all register to the same multicast address. Any application (whether registered to the multicast address or not) can communicate to all the registered applications by sending packets to the multicast address. The multicast mode is preferable if multiple applications need to receive the same information.

Node Communication in Guidewire Clusters

Guidewire application clusters rely on a mix of multicast and unicast communication:

- Each node, at startup, reads the unicast address of the current coordinator from the database. The startup node then initiates a JGroups message exchange with the coordinator to join the cluster.
- After joining the cluster, the startup node runs a checksum on its own configuration. The node then sends a unicast message to the cluster coordinator and compares its checksum with that of the coordinator. If the startup node detects that the two checksums are not equal, it refuses to start. This prevents the start of different version levels of the application or configuration on the same database.
- If a new startup node has the `isbatchserver` value set to true, it reads the database to determine if there is a running batch server already. The startup node then makes multiple attempts to send a unicast request to the currently listed batch server. If current batch server responds, the new node start up fails, as there is already a batch server. If the existing batch server does not respond, then the new node starts successfully and overwrites the batch server record in the database.

- At regular intervals, each server sends unicast pings to the other servers in the cluster to detect server failure and to maintain an up-to-date cluster view.
- At times, a non-batch node needs to communicate directly with the batch server node to forward a request that the batch server alone can process. BillingCenter clusters store the batch server address and server ID in the database. If a cluster node wants to communicate with the batch server, the node reads the information on the current batch server from the database. A direct communication between the requesting server and the batch server then takes place through unicast requests and responses.
- Cache eviction messages constitute the largest portion of cluster communication. Guidewire applications leverage an internal cache to minimize reads to the database. If a node changes a cache entry, that node sends a cache eviction message on the multicast channel to all other nodes in the cluster. This message notifies all the other nodes to invalidate that cache entry, if they have it. After invalidating the cache entry, on the next read, a node reads the corresponding data directly from the database. See “Cache Usage in Guidewire Clusters” on page 77 for more details.

Cache Usage in Guidewire Clusters

Cluster inter-communication ensures that if an object changes in one node cache, that node sends a cache invalidation message to other nodes in the cluster. This message instructs the other nodes to tag the cache entry for the object as obsolete and evict it from the cache. The next time a server needs the object, the server 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 nodes.

Multicast is a non-guaranteed transport. Thus, it is possible to lose message packets. Network failures or other issues also can disrupt communication between nodes. Such cases can result in cache eviction messages not being propagated to all nodes. As a result, one or more nodes 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 the entities were last accessed. This mismatch results in BillingCenter 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

- “Application Server Caching” on page 63
- “Cache Management” on page 63
- “Concurrent Data Change Prevention” on page 64

Configuring a Cluster

In general, to create and configure a BillingCenter cluster, you need perform the following steps:

- First, install the BillingCenter server on all the nodes in the cluster. Install the BillingCenter application server in the same way that you install a standalone BillingCenter server. If you install multiple BillingCenter application server on the same machine, each BillingCenter application server must run in its own JVM instance.
- Configure the individual servers in the cluster, changing each server’s configuration settings as appropriate for that server. Use the information in “Configuring Individual Cluster Servers” on page 79 to determine how to configure the individual cluster members.

This topic discusses various aspects of configuring a clustered installation. This topic includes the following:

- “List of Cluster Configuration Parameters” on page 78
- “Configuring Individual Cluster Servers” on page 79

- "Enabling and Disabling Clustering" on page 79
- "Configuring the Registry Element for Clustering" on page 79
- "Setting the Multicast Address" on page 81
- "Specifying the Key Range" on page 82
- "Configuring Separate Logging Environments" on page 82

See also

- "Considerations for a Clustered Application Server Environment" on page 16 in the *Installation Guide*
- "Using TCP Instead of UDP" on page 16 in the *Installation Guide*
- "Disabling IPv6 in Clustered Environments" on page 17 in the *Installation Guide*

List of Cluster Configuration Parameters

The following table lists the configuration parameters associated with clustering.

Parameter	Default	Description
`ClusteringEnabled`	False	Whether to enable clustering on this application server.
`ClusterMemberPurgeDaysOld`	30	Number of days to keep cluster member records before purging.
`ClusterMemberRecordUpdateIntervalSecs`	60	Cluster member database record update interval (in seconds). Unused in JGroups-based server clusters.
`ClusterMulticastAddress`	228.9.9.9	IP multicast address to use in communicating with the other members of the cluster. Unused if `ClusteringEnabled` is false.
`ClusterMulticastPort`	45566	Port used to communicate with other members of the cluster. Unused if `ClusteringEnabled` is false.
`ClusterMulticastTTL`	1	Time-to-live for cluster multicast packets. Unused if `ClusteringEnabled` is false.
ClusterProtocolStack ClusterProtocolStackOption1 ClusterProtocolStackOption2	-	JGroups configuration template to use with multicast UDP communications. See "ClusterProtocolStack" on page 37 in the *Configuration Guide* for details.
`ClusterStatisticsMonitorIntervalMins`	60	Number of minutes between each cluster statistics monitor logging. Unused in JGroups-based server clusters.
`ConfigVerificationEnabled`	True	Whether to check the configuration of this server against the other servers in the cluster. **Guidewire specifically does not support disabling this parameter in a production environment.**
`JGroupsClusterChannel`	True	Whether to use JGroups-based cluster channel.
`JGroupsWatchdogHeartbeatIntervalSecs`	30	Number of seconds between each heartbeat ping from the cluster coordinator to each of the nodes.
`JGroupsWatchdogMissedHeartbeatsBeforeReset`	10	Number of missed heartbeats after which a node resets its JGroups channel.
`PDFMergeHandlerLicenseKey`	-	Provides license key for server-side PDF generation. It is possible to set this value differently for each server node in cluster.

For a discussion of configuration parameters in general, see "Application Configuration Parameters" on page 27 in the *Configuration Guide*. For a discussion of cluster-specific configuration parameters, see "Clustering Parameters" on page 35 in the *Configuration Guide*.

Configuring Individual Cluster Servers

With some exceptions, all the servers in a cluster must have identical `config.xml` files and identical metadata. There are directories within the `configuration` module (`BillingCenter/modules/configuration`) that need not be identical among clustered servers, these directories are:

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

As each server starts, it connects to an existing server in the cluster and compares its configuration environments.

- If the configurations between the two servers differ, the new server fails to start.
- If the new server has the same server ID as any other server in the cluster, it refuses to start and does not join the cluster.

To configure a cluster, you use several parameters in the `config.xml` file and make use of the BillingCenter environment properties to ensure that server-specific properties resolve correctly. See the following sections for details:

- “Enabling and Disabling Clustering” on page 79
- “Configuring the Registry Element for Clustering” on page 79
- “Setting the Multicast Address” on page 81
- “Specifying the Key Range” on page 82
- “Configuring Separate Logging Environments” on page 82

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

Enabling and Disabling Clustering

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

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

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

See also

- “List of Cluster Configuration Parameters” on page 78
- “Clustering Parameters” on page 35 in the *Configuration Guide*

Configuring the Registry Element for Clustering

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

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

```
</registry>
```

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

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

Developing Sophisticated Configurations by Using Localized Parameters

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

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

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

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

At startup, BillingCenter first resolves the `env` element. Assuming that `-Dgw.bc.env` is not set on the command prompt to another value, this `registry` would result in the `env` resolving to `cluster1`. With this configuration, the server that starts first, becomes the batch server.

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

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

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

Defining a Batch Server with the isbatchserver Environment Property

A batch server performs the following operations:

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

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

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

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

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

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

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

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

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

To use the `-D` option to set a system property at the server start, use the form `-Dgw.bc.`*`SystemProperty`*, with *`SystemProperty`* being the name of the system property. For example, to set the `isbatchserver` system property at the start of a development server, use the following command:

```
gwbc dev-start -Dgw.bc.isbatchserver="true"
```

For a complete discussion of how environment properties work, see “Defining the Application Server Environment” on page 14. Also, see the example in creating this system property in “Adding a Server to a Cluster” on page 83.

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

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

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

   is the same as:

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

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

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

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

Setting the Multicast Address

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

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

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

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

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

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

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

224.0.0.0 – 239.255.255.255

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

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

Therefore, choose a multicast address in the following range:

224.0.0.0 – 239.255.255.255

If a cluster contains several systems that all share the same subnets, it is important that the systems use different multi-cast I.P addresses. Thus, if multiple Guidewire applications all operate on the same network, each Guidewire application needs to use a different dedicated multicast IP address to avoid conflicts.

Specifying the Key Range

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

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

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

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

Configuring Separate Logging Environments

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

You can also specify unique logging files per server within the same environment. See “Configuring Logging in a Multiple Instance Environment” on page 28.

Managing a Cluster

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

Starting Clustered Servers

To start BillingCenter servers in a cluster

1. Start the batch server.
2. Wait until you see the following in the log for the batch server:
   ```
   date time INFO ***** BillingCenter ready *****
   ```
3. Start web service BillingCenter servers. After executing each server start, wait ten seconds. Then, start the next web service BillingCenter server.
4. On each web service BillingCenter server, check that the server has started by testing the server. Direct a browser to the BillingCenter HTTP ping utility:
   ```
   http://server:port/bc/ping
   ```
   When the server is running at the default `MULTIUSER` run level, the browser displays the number `2`. For more information on the BillingCenter HTTP ping utility, see “Checking Node Health” on page 86.
5. For each web service server, invoke `http://server:port/bc/soap/someWSAPI?WSDL` where *someWSAPI* is a web service that you have defined. This publishes all WSDLs and ensures that web services are ready.
6. Start regular BillingCenter servers for handling user requests. After executing each server start, wait ten seconds. Then, start the next BillingCenter server.

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

Checking Server Run Level

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

```
http://server:port/bc/ping
```

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

Code	Run level
2	MULTIUSER
-	DAEMONS
(	MAINTENANCE

Also see “Checking Node Health” on page 86.

Adding a Server to a Cluster

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

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

To add a server to the cluster

1. On the server you want to add, remove the `config` directory and the `config.xml` file.
2. Copy the `config` directory and the `config.xml` file from a server on the cluster to the server you want to add.
3. If using the `config.xml` cluster registry, set the `serverid` registry element to a unique value within the cluster. Otherwise, the server with the duplicate ID refuses to start and does not join the cluster. See "Defining the Application Server Environment" on page 14.
4. Start the new server.

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

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

Adding a Server to an Application Cluster Dynamically

BillingCenter does not require that you use the cluster registry in file `config.xml` to define cluster members. Instead, it is possible to specify the server `serverid` and the `isBatchServer` flag using command prompt arguments as you start the server. Or, you can specify the batch server only in the cluster registry in file `config.xml` and set the `serverid` value as you start each server. In this way, you do not need to modify the `config` file of all cluster members and restart each one to add a new server to the cluster.

If you want to start a cluster member with the `isBatchServer` flag, you first need to create a system property of type `isBatchServer` in `config.xml`. See "Defining a Batch Server with the isbatchserver Environment Property" on page 80 for additional discussion of system properties.

The following example shows how to define a system property of type `isbatchserver` to use as an `gw.bc.isbatchserver` option:

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

To use this system property with a development server, enter something similar at the command prompt (using the example system property):

```
gwbc dev-start -Dgw.bc.isbatchserver="true"
```

For this system property to be useful, you must create the system property as part of your initial cluster configuration and it must exist on all cluster members. In that way, it is available to use dynamically as required.

You can also combine the `isbatchserver` property with adding a server (with name *`ServerName`*) to the cluster dynamically by setting the `serverid` value, for example:

```
gwbc dev-start -Dgw.bc.serverid="ServerName" -Dgw.bc.isbatchserver="true"
```

Removing a Server from a Cluster

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

To remove a server from the cluster

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

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

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

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

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

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

Running Administrative Commands

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

Updating Clustered Servers

To update a server, shut it down, deploy an updated application file, and start the server again. For more information, see “Deploying BillingCenter to the Application Server” on page 79 in the *Installation Guide*.

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

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

Monitoring Cluster Health

The following topics describe ways in which you can monitor the health of Guidewire cluster:

Using the Cluster Info Page

The **Cluster Info** page, accessible to system administrators, provide a user interface that displays cluster information, if clustering is enabled. This information includes:

- Information on this node's application server
- Information on the batch server
- Information on other members of the cluster
- History of each member in the cluster.

From this page, you can make the current node the batch server, refresh cluster information, or generate a report on the cluster.

See also

- "Cluster Info" on page 173

Checking Node Health

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

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

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

```
http://localhost:8080/bc/ping
```

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

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

Guidewire based this HTTP ping utility on a system run level checker built for developers. See "Getting and Setting the Run Level" on page 129 in the *Integration Guide*. Invoking this utility programmatically provides more granular information on the server's status.

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

chapter 7

Securing BillingCenter Communications

Guidewire products use a standard three-tier architecture:

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

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

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

This topic includes:

Using SSL with BillingCenter

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

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

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

Overview of the Steps

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

1. “Editing the httpd.conf File” on page 88
2. “Editing the httpd-ssl.conf File” on page 88
3. “Editing the server.xml File” on page 89

Editing the httpd.conf File

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

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

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

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

Editing the httpd-ssl.conf File

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

To edit the Apache SSL module

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

   ```
   Listen 443
   ```

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

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

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

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

      #Access is allowed to the bc8.0.4 and
      #its subdirectories for the authorized sites only
      <Directory /bc8.0.4>
            Order Deny,Allow
            Allow from all

            # Never allow communications to be not encrypted
            SSLRequireSSL

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

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

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

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

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

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

5. Start Apache.

   net start Apache2

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

Editing the server.xml File

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

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

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

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

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

Specify the following:

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

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

Accessing a BillingCenter Server Through SSL

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

`https://server:port/bc/BillingCenter.do`

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

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

Handling Browser Security Warnings

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

To disable this warning

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

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

chapter 8

Importing and Exporting Administrative Data

Guidewire categorizes certain types of application data as administration data. (Frequently, the term is shortened to just *admin data*.) For example, activity patterns are administration data. This topic provides information related to importing administrative data into BillingCenter. While users enter much of the information into BillingCenter directly, at times, it is more convenient or necessary to enter information in bulk.

This topic discusses how to export administrative data as well.

This topic includes:

Ways to Import Administrative Data

It is possible to import administrative data into Guidewire BillingCenter using the following mechanisms.

Mechanism	How	Description
BillingCenter	**Administration** tab	Import and export administrative data files in XML format using functionality available from the BillingCenter **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 BillingCenter" on page 101
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 187 • "Importing Administrative Data Using the import_tools Command" on page 100 • "Importing Administrative Data" on page 127 in the *Integration Guide*
File load	`import` directory	Load data from CSV files contained in directory **configuration** → **config** → **import** → **gen**. At initial database upgrade, starting from an empty database, BillingCenter loads the administrative data contained in the files in this directory. This includes the default activity patterns, authority limits, and roles and role privileges. You can add additional files to this directory for load at initial server startup. For details, see "Understanding the import Directory" on page 92. **IMPORTANT** BillingCenter loads files from the **import** → **gen** directory only if the database is empty, during the database upgrade at initial server start.
Staging tables	`table_import` `TableImportAPI`	Use the `table_import` command prompt tool to import administrative data from staging tables into BillingCenter database tables. It is also possible to use the `TableImportAPI` web service to import administrative data from staging tables into BillingCenter database tables. It is possible to use these tools to import data of any type into the database. For more information, see: • "Table Import Command" on page 195 • *"Importing from Database Staging Tables" on page 367 in the Integration Guide.*

Understanding the import Directory

After the initial BillingCenter installation, the database is empty of data. The first time that you start the BillingCenter application server after installation, BillingCenter upgrades the database and populates it with data. As part of this initial upgrade, BillingCenter 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)

BillingCenter 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**

The actual files can be empty or modified to meet your business needs. However, the file must exist or the database upgrade fails.

BillingCenter also examines file `importfiles.txt` (in the same **gen** location) to determine if it lists additional administrative data files to load. In the base configuration, file `importfiles.txt` contains the name of one file to load, that of `authority-limits.csv`.

If you wish to load additional administrative data at initial database upgrade, do the following:

1. Place the appropriately formatted CSV files containing the data in the **gen** folder in Studio. The file format must be CSV.
2. Add the filename to the list of files in `importfiles.txt`. Any file named in `importfiles.txt` must exist in the **gen** folder or the database upgrade fails.

BillingCenter loads the data in sequential order of the files listed in file `importfiles.txt`.

Re-importing Administrative Data in the gen Folder

In general, it is not possible to import additional administrative data contained in the **gen** folder after the initial database upgrade. BillingCenter loads the administrative data contained in the **gen** folder only if starting from an empty (non-upgraded) database. Thereafter, BillingCenter ignores the files in the **gen** directory, with the following exception.

It is possible, however, to re-import the data in file `roleprivleges.csv` in the **gen** folder by using the `-privileges` option of the `import_tools` command. This command option rebuilds the privileges associated with each BillingCenter role by doing the following:

- It first deletes the existing role privileges in the current database
- It then re-imports the contents of the `roleprivileges.csv` file in the **gen** folder.

See “Import Tools Command” on page 187 for more details.

Setting the Character Set Encoding for File Import

In the base configuration, if you do not supply a value for the `import_tools -charset` option, BillingCenter 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 “Import Tools Options” on page 188 for more information.

Maintaining Data Integrity During Administrative Data Import

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

To maintain data integrity while importing data

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

Administrative Data and the BillingCenter Data Model

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

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

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

```
gwbc regen-dictionary
```

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

See also

- "Working with the Data Dictionary" on page 141 in the *Configuration Guide*

Public ID Prefix

Each entity that you import into BillingCenter requires a unique public ID. This is separate from the system ID that BillingCenter 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 could have a contact entity with ID=5432. Thus, Guidewire recommends that you use the following ID format to ensure that the IDs do not register as duplicates:

```
origin:ID
```

By using this format, the contact from the first system comes in as `adminsystem:5432` and the contact from the second system comes in as `salessystem:5432`. 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. Public IDs cannot exceed 20 characters in length.

Support for Unique Public IDs in a Development Environment

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

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

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

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

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

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

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

Constructing a CSV File for Import

You can only import data for an entity 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 and data information. BillingCenter treats any line that contains the string `entityid` as a heading. BillingCenter considers as data any line:

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

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

IMPORTANT Guidewire supports using the `import_tools` command to import administrative data only.

Constructing a Heading Line

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

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

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

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

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

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

```
1: type,data-set,entityid,firstname,lastname,primaryaddress,
  workphone,primaryphone,taxid,vendortype,specialtytype
2: Person,0,demo_sample:1,Ray,Newton,demo_sample:4000,818-446-1206,work,,,
3: Person,0,demo_sample:2,Stan,Newton,demo_sample:4002,818-446-1206,work,,,
```

```
4: Person,0,demo_sample:3,Harry,Shapiro,demo_sample:1004,818-252-2546,work,,,
5: Person,0,demo_sample:4,Bo,Simpson,demo_sample:1003,619-275-2346,work,,,
6: Person,0,demo_sample:5,Jane,Collins,demo_sample:4003,213-457-6378,work,,,
7: Person,0,demo_sample:6,John,Dempsey,demo_sample:1006,213-475-9465,work,,,
```

The `primaryaddress` field is a foreign key to the `Address` entity. It appears as `PrimaryAddressID` in the *BillingCenter 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

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 *BillingCenter 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,,,,,,,,,,
```

BillingCenter 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, the data-set of 10240 is requested. BillingCenter assumes that data-set 10240 includes every data-set that might be created in practice. The second field can be left blank, in which case BillingCenter always includes this object in the import regardless of which data-set is requested.

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

Simple Array Data

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

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

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

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

Complex Array Data

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

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

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

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

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

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

Constructing an XML File for Import

To create a properly formatted XML file of administrative data for import, Guidewire recommends that you do the following:

1. First, export the current administrative data as an XML file, using the functionality available on the **Export Data** screen available to BillingCenter 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 with the following command, from the BillingCenter installation `bin` directory:

   ```
   gwbc regen-xsd
   ```

   It is important to regenerate the XSD files every time that you modify the data model.
4. Re-import the modified file using either the `import_tools` command or the **Export Data** screen available to BillingCenter administrators. This screen provides the ability to import administrative data in XML format into BillingCenter.

For information on how to perform each one of these steps, see the following:

- “Importing and Exporting Administrative Data from BillingCenter” on page 101
- “Constructing the XML for the Administrative Data Import File” on page 98
- “Import Tools Command” on page 187

IMPORTANT Guidewire supports using the `import_tools` command to import administrative data only.

Constructing the XML for the Administrative Data Import File

The `BillingCenter/build/xsd/bc_import.xsd` file defines the XML schema used for import and export. This file further references information in two other XSD files in the same directory:

- `bc_entities.xsd`
- `bc_typelists.xsd`

You can use any schema-aware XML editor to help format information properly according to these XSD definitions. You generate these XSD files with the following command:

```
gwcc regen-xsd
```

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

The following XML example shows the default activity pattern **30 Day Diary** from the BillingCenter 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 using the existing entries as a model and re-import the file.

Foreign Key References

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

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

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

Within a single XML file you can reference an item before defining the item. This enables you to define all of the groups first, for example, including referring to supervisor users who are not defined until later in the file. BillingCenter 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 BillingCenter XSD files any time you modify the data model. These files are likely to change as you configure the data model. You generate these XSD files with the following command from the BillingCenter installation `bin` directory:

```
gwbc regen-xsd
```

You can validate the XML of your import file against an `bc_import.xsd` file 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 bc_import.xsd
var schemaFile = new File("bc_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())
```

Importing Administrative Data Using the import_tools Command

Guidewire provides an import tool for importing new or updated administrative data into existing tables in the BillingCenter database. BillingCenter reads the import data from a CSV (comma-separated values) file or an XML file. *BillingCenter supports this command for importing administrative data but not for importing other types of data.* Instead, use staging tables or APIs to import other types of data into BillingCenter. See “Import Tools Command” on page 187 for more information on the `import_tools` utility.

BillingCenter 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 BillingCenter Data Model” on page 94 for a discussion of public IDs

During import, if BillingCenter finds a match in entity public IDs, it does the following:

- If there is no difference between the import record and the database record, BillingCenter ignores the import record.
- If there are differences between the two records, BillingCenter overwrites any existing database record values with the values from the import file.
- If there are null entries for a record in the import file, BillingCenter nulls out those values in the record in the database.

IMPORTANT Guidewire supports using the `import_tools` command to import administrative data only.

To import administrative data using the import_tools command

1. Create a CSV or XML file describing the data, using one of the following methods:
   - Create the XML or CSV file manually.
   - Export the current administrative data as an XML file from BillingCenter and modify the file.
2. Import the CSV or XML file using the `import_tools` command, for example:

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

The `import_tools` command requires that you supply a password (*`password`*). 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.

Note: 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). In the base configuration, the default value of `MaximumFileUploadSize` is 20 MB.

See also

- “Constructing a CSV File for Import” on page 95
- “Constructing an XML File for Import” on page 98
- “Import Tools Command” on page 187

Importing and Exporting Administrative Data from BillingCenter

You can import and export administrative data directly from BillingCenter using functionality that is available to BillingCenter administrators. It is only possible to import and export XML-formatted administrative data through the BillingCenter interface.

Importing Administrative Data into BillingCenter

You can import administrative data directly from within BillingCenter using the **Import Data** screen available to BillingCenter administrators. To access this screen, navigate to the following location in BillingCenter:

Administration → **Utilities** → **Import Data**

You can import XML-formatted administrative data only.

During the import, BillingCenter looks for existing data objects that have public IDs that match those of the data objects being imported. BillingCenter 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

To import administrative data from BillingCenter

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

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

Importing Arrays

BillingCenter handles arrays differently depending on whether it is importing an owned array or a virtual array. If an entity owns the array, BillingCenter 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. BillingCenter only gives you the option to delete the current array and replace all of the contents of the array. Virtual arrays, because they cannot be deleted, cannot be replaced by an import at all.

Exporting Administrative Data from BillingCenter

You can export administrative data directly from within BillingCenter using the **Export Data** screen available to BillingCenter administrators. To access this screen, navigate to the following location in BillingCenter:

Administration → **Utilities** → **Export Data**

You can export the following data sets independently:

- Admin
- Holidays
- Roles

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

You can export XML-formatted administrative data only.

To export administrative data from BillingCenter

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

Importing Roles and Permissions

A *permission* (or privilege) is a granular task or ability to see or do something within BillingCenter. A *role* is a named collection of permissions, and, typically, maps to a job function or job title.

BillingCenter stores role information in file `roles.csv` and permission information in file `roleprivileges.csv`. Within Guidewire Studio, these two files exist in the following location:

configuration → **config** → **import** → **gen**

BillingCenter loads the contents of these two files into the database upon initial database upgrade, at first server startup after installation. See "Understanding the import Directory" on page 92 for details on how BillingCenter works with the files in the **gen** directory.

Re-importing Administrative Data in the gen Folder

In general, it is not possible to import additional administrative data contained in the **gen** folder after the initial database upgrade. BillingCenter loads the administrative data contained in the **gen** folder only if starting from an empty (non-upgraded) database. Thereafter, BillingCenter ignores the files in the **gen** directory, with the following exception.

It is possible, however, to re-import the data in file `roleprivleges.csv` in the **gen** folder by using the `-privileges` option of the `import_tools` command. This command option rebuilds the privileges associated with each BillingCenter role by doing the following:

- It first deletes the existing role privileges in the current database
- It then re-imports the contents of the `roleprivileges.csv` file in the **gen** folder.

See "Import Tools Command" on page 187 for more details.

See also

- "Ways to Import Administrative Data" on page 92
- "Users, Groups, and Roles" on page 383 in the *Application Guide*

Role Definitions

File `roles.csv` contains a list of BillingCenter 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. BillingCenter reads the file, starting with the first row that contains the `entityid` identifier and imports the data into the database.

The following are examples of role definition entries:

```
Roles,
type,data-set,entityid,description,name,carrierinternalrole
Role,0,rule_admin,Permissions for rule admin,Rule Admin,TRUE
Role,0,superuser,Superuser with full permissions,Superuser,TRUE
```

```
Role,0,user_admin,Permissions for user admin,User Admin,TRUE
Role,0,underwriter,Permissions for underwriter,Underwriter,TRUE
,,,,,
```

Role Permission Definitions

File `roleprivileges.csv` contains the mappings that link roles to a set of permissions. BillingCenter reads the file starting with the first row that contains the `entityid` identifier and imports the data into the database.

The following are examples of permission definition entries:

```
RolePrivileges
type,data-set,entityid,permission,role
RolePrivilege,0,default_data:1,acctchargesview,billing_clerical
RolePrivilege,0,default_data:2,acctchargesview,billing_manager
RolePrivilege,0,default_data:3,acctchargesview,commissions_admin
RolePrivilege,0,default_data:4,acctchargesview,finance_clerical
RolePrivilege,0,default_data:5,acctchargesview,finance_manager
,,,,
```

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 `acctchargesview` entry grants permission to view account charges to the `billing_clerical` role.

A full list of permissions, along with a brief description of each, is listed in the *BillingCenter Data Dictionary.* See the `SystemPermissionType` typelist. See the *BillingCenter Security Dictionary* for a list of the correspondences between roles and permissions.

Importing Authority Limits and Authority Limit Profiles

An *authority limit profile* is a named collection of authority limits. As the name suggests, an *authority limit* provides limits or restrictions on the size of a reserve or a reserve increase that a user can make. It can also limit the maximum size of a payment a user can authorize.

BillingCenter stores information on authority limits and authority limit profiles in file `authority-limits.csv`. Within Guidewire Studio, this file exists in the following location:

configuration → **config** → **import** → **gen**

BillingCenter loads the contents of this file into the database upon initial database upgrade, at first server startup after installation. See “Understanding the import Directory” on page 92 for details on how BillingCenter works with the files in the **gen** directory.

IMPORTANT It is not possible to add additional administrative data through this mechanism after the initial database upgrade. BillingCenter loads the administrative data contained in the **gen** folder only if starting from an empty (non-upgraded) database.

See also

- “Ways to Import Administrative Data” on page 92
- “Authority Limits and Authority Limit Profiles” on page 386 in the *Application Guide*

Authority Limit Definition

File `authority-limits.csv` consists of two sets of data:

- `AuthorityLimitProfile` rows
- `AuthorityLimit` rows

Each authority limit profile definition contains a human-readable name and description for each role. Within the profile definition, set the `name` and `description` fields to whatever is useful in uniquely identifying the profile.

In `authority-limits.csv`:

- BillingCenter reads the file, starting with the first row that contains an `entityid` identifier and imports that data into the database as profile data.
- BillingCenter continues reading the file until finding the second `entityid` identifier, at which point, BillingCenter imports the remaining data into the database as authority limit data.

For example:

```
Authority Limit Profiles,
type,data-set,entityid,name,description
AuthorityLimitProfile,0,default_data:1,Superuser,Superuser default authority
AuthorityLimitProfile,0,default_data:2,General,General default authority
AuthorityLimitProfile,0,default_data:3,Low,Low default authority

type,data-set,entityid,profile,limittype,limitamount
AuthorityLimit,0,default_data:1,default_data:1,accountcreditcollections,1000000 usd
AuthorityLimit,0,default_data:2,default_data:1,accountcreditcollectionsreversal,1000000 usd
AuthorityLimit,0,default_data:3,default_data:1,accountcreditgoodwill,1000000 usd
AuthorityLimit,0,default_data:4,default_data:1,accountcreditgoodwillreversal,1000000 usd
AuthorityLimit,0,default_data:5,default_data:1,accountcreditinterest,1000000 usd
...
```

The definitions in this example give the `General` profile (`default_data:2`) a $1,000,000 (USD) credit collections reversal limit.

See the *BillingCenter Data Dictionary* for the `AuthorityLimitType` typelist for the possible limit type codes (for example, `ctr`). The `AuthorityLimitType` typelist contains the available codes. See the *BillingCenter Data Dictionary* for details.

Importing Security Zones

In the base default configuration, Guidewire provides a single security zone named Default Security Zone. It is possible to add additional security zones in BillingCenter through the following methods:

- By defining a new security zone using the **Security Zones** screen in the BillingCenter **Administration** tab. See “Security Zones” on page 388 in the *Application Guide* for details.
- By importing new security zone data using the export and import functionality accessible in the BillingCenter **Administration** tab. See the following procedure for details.

To import new security zones

1. Within BillingCenter, 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 94 for more information on public IDs.

5. Add unique entries for each security zone to import. Security zone elements have the following form:

   ```
   <SecurityZone public-id="bc:236">
     <description>Some meaningful description...</description>
     <name>Some meaningful name...</name>
   </SecurityZone>
   ```

   This example sets the `public-id` value to `bc:236`. Chose a public ID value that makes business sense for your organization.

6. After saving the file, navigate to the following location in BillingCenter:

   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, BillingCenter 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**.

You can now view and use the updated set of security zones in the **Administration** → **Security Zones** screen without application server restart.

Importing Zone Data

BillingCenter provides a collection of zone data files for various localities with small sets of zone data that you can load for development and testing purposes. The zone data files are in the following location in the Studio **Project** window:

configuration → **config** → **geodata**

Within the **geodata** folder, BillingCenter 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. For example, the provided zone data files are not complete and may not include recent changes.

Also, the formatting of individual data items in these files might not conform to your internal standards or the standards of third-party vendors that you use. For example, the names of streets and cities are formatted with mixed case letters but your standards may require all upper case letters.

The `US-Locations.txt` file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings. You can edit the `US-Locations.txt` file to conform to your particular address standards, and then import that version of the file.

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.

Before you use the `zone_import` command, set the BillingCenter server run level to `MAINTENANCE`. After you use the command to import zone date, move the data from the staging tables to the production tables by using the `table_import` command. See “Table Import Command” on page 195 for more information on the `table_import` command.

Guidewire expects that you import address zone data upon first installing BillingCenter, and then at infrequent intervals thereafter as you receive data updates.

To import a zone data file

1. Start the BillingCenter server.

2. Set the BillingCenter server run level to `MAINTENANCE`:

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

3. Clear the zone data staging tables. If you have multiple countries defined, you can include the `-country countryCode` option to clear staging zone data only for the country you will be loading:

   ```
   zone_import -password password -clearstaging [-country countrycode]
   ```

4. Load the zone data file into the staging tables:

   ```
   zone_import -password password -country countrycode -import filename
   ```

5. Clear existing zone data in production. This is useful if there is already zone data for the particular country that you are loading:

   ```
   zone_import -password password -country countrycode -clearproduction
   ```

6. Load zone data from the staging tables into production:

   ```
   table_import -password password -integritycheckandload -zonedataonly
   ```

7. Set the server run level back to `MULTIUSER`:

   ```
   system_tools -password password -multiuser
   ```

See also

- For information on using the `zone_import` command, see "Zone Import Command" on page 197.
- For information on using the `table_import` command, see "Table Import Command" on page 195.
- For more information on database staging tables, see "Importing from Database Staging Tables" on page 367 in the *Integration Guide*.
- For information on the web service `ZoneImportAPI` that also imports zone data, see "Introduction to Database Staging Table Import" on page 367 in 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. The `zone-config.xml` files for specific countries are located in folders for specific countries. Navigate in the Studio **Project** window to **configuration** → **config** → **geodata**.

The following example is the zone configuration file for zone data for France.

configuration → **config** → **geodata** → **FR** → `zone-config.xml`

Each `zone-config.xml` files must have one `Zones` element for a specific country. The `Zone` element defines the fields in zone data files for a country. For each field, the `fileColumn` attribute indicates the position of the field within lines of the files.

The following example XML code from a `zone-config.xml` file defines the fields to import from zone data files for the United States. The example code specifies for United States zone data files that the third field in the comma-separated values of each line of corresponds to a city.

```
<Zones countryCode="US">
  ...
  <Zone code="city" fileColumn="3" granularity="2">
  ...
```

See also

- For complete information about `zone-config.xml`, including a description of its XML elements and attributes, see "Configuring Zone Information" on page 126 in the *Globalization Guide*.

chapter 9

Batch Processing

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

This topic includes:

See also

- "Custom Batch Processing" on page 407 in the *Integration Guide*

Overview of Batch Processing

BillingCenter supports two styles of batch processing:

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

  A work queue comprises the following components:

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

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

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

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

Work Queues

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

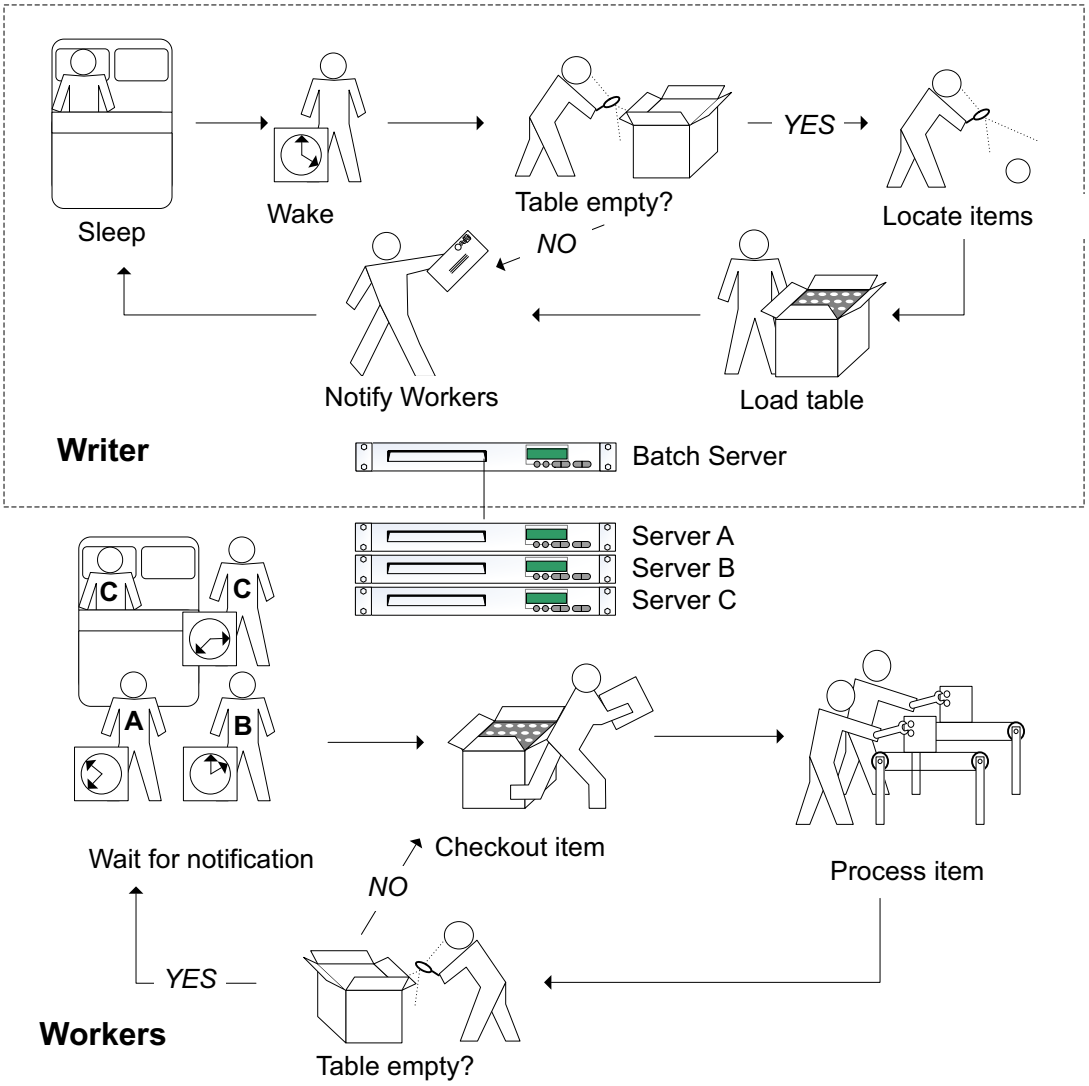
A work queue comprises the following components:

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

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

Work Queue Architecture

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



Functions of the Writer

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

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

1. Notifies the workers that they have work to process.
2. The writer thread ends.

If no work items are found, the writer thread:

1. Selects items for a new batch.
2. Writes the identifies of the selected items to the work queue table.
3. Notifies the workers that they have work to process.
4. The writer thread ends.

Functions of a Worker

Typically, work queues share the standard work item table for their work items. However, a worker thread operates only on work items in that table inserted by its associated writer. For example, the Activity Escalation work queue might be configured for six workers on three different servers in a BillingCenter cluster. Those workers work only on activity escalation items in the standard work item table. Typically, you configure work queues for multiple worker threads, so typically some number of worker threads are operating throughout the day on items in the standard work queue table.

Whenever a worker thread awakens, it checks the work item table for work items from its associated writer. Sometimes workers are notified of work but find none available when they awaken. For small batch runs, a worker can check out all items in the batch with its first quota 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.

If a worker awakens and finds work items available for processing, the worker checks out its quota from the work queue. For each item, the worker sets the following attributes.

`Status`	Set to `checkedout`. This attribute can be `available`, `checkedout`, or `failed`.
`LastUpdateTime`	Set to the time when the worker checks out the work item.
`CheckedOutBy`	Set to the worker.

After it checks out a quota of work items, the worker thread processes them sequentially. Whenever a worker completes a work item successfully, it deletes the item from the table and begins to process the next item. The standard work item table (`StandardWorkQueue`) is retireable, so successfully completed work items remain in the table for historical reference.

Work Queue Scheduling and Processing Intervals

A writer for a work queue starts at the interval specified in `scheduler-config.xml`. Typically, you schedule the writer to start several times during the day or once at night. The writer thread runs on the batch server, just like a batch process. Access the schedule configuration file `schedule-config.xml` in the **Project** window of Guidewire Studio at **configuration** → **config** → **scheduler**.

Worker threads awaken much more frequently than their writers start. One worker awakens at least every `maxpollinterval` if not more frequently. You do not schedule worker threads. Instead, they awaken in response to notification from the writer or upon expiry of the polling interval. After a worker awakens, if there are work items to process, it processes up to `batchsize` items. If there are more items than the batch size to process, the worker awakens another worker. This worker repeats the process of checking out work items and waking up another worker if necessary, until the configured number of workers is reached. You configure the number of workers, polling interval, and batch size in `work-queue.xml`. Access the work queue configuration file `work-queue.xml` in the **Project** window of Guidewire Studio at **configuration** → **config** → **workqueue**.

See also

- “Scheduling Work Queue Writers and Batch Processes” on page 113
- “Performing Custom Actions After Batch Processing Completion” on page 118

Batch Processes

Batch processes execute on the batch server only. Generally, a batch process runs to completion and then reports its result back to a log or to the administrative user interface. You can view and manage batch processing from the **Batch Process** page on the **Server Tools** tab in the BillingCenter application.

See also

- “Running Work Queue Writers and Batch Processes” on page 111

Running Work Queue Writers and Batch Processes

You can run many batch processes, including writers for work queues, from BillingCenter or from the command prompt.

This topic includes:

See also

- "Running Batch Processes Using Web Services" on page 128 in the *Integration Guide*

Running a Writer from BillingCenter

You can run the writer for a work queue from the **Work Queue Info** screen. From the **Work Queue Info** screen, you can see the progress of workers processing items in the work queue. The process history that you download from the **Work Queue Info** screen includes history for the run of the writer, including its duration and starting and ending times.

Users must have the `internaltools` permission to access the **Work Queue Info** page. The Admin role has this permission by default. Alternatively, if the `EnableInternalDebugTools` parameter is set to true in `config.xml` and the server is running in development mode, all users can access the **Work Queue Info** page.

To run the writer for a work queue

1. Log in to BillingCenter.
2. Press `ALT+SHIFT+T` to display the **Server Tools** tab.
3. Click **Work Queue Info** in the left sidebar.
4. Click **Run Writer** in the **Action** column of the work queue. The **Run Writer** button is enabled for all work queue types that belong to the `BatchProcessTypeUsage` category `UIRunnable`.

See also

- "Work Queue Info" on page 157
- "Server Modes and Run Levels" on page 56

Running a Batch Process from BillingCenter

You can run batch processes from the **Batch Process Info** screen in BillingCenter. The **Batch Process Info** screen also contains information such as the current status of the batch process, when it last ran, when it will run again, and the schedule.

Note: You can run writers for work queues from the **Work Queue info** page or the **Batch Process Info** page.

Users must have the `internaltools` permission to access the **Batch Process Info** page. The Admin role has this permission by default. Alternatively, if the `EnableInternalDebugTools` parameter is set to true in `config.xml` and the server is running in development mode, all users can access to the **Batch Process Info** page.

To run a batch process from the Batch Process Info page

1. Log in to BillingCenter.

2. Press ALT+SHIFT+T to display the **Server Tools** tab.
3. Click **Batch Process Info** in the left sidebar.
4. Click **Run** in the **Action** column of the batch process that you want to run. The **Run** button is enabled for all batch process types that belong to the `BatchProcessTypeUsage` category `UIRunnable`.

See also

- "Batch Process Info" on page 156
- "Server Modes and Run Levels" on page 56

Running a Writer or Batch Process from the Command Prompt

You can run many batch processes, including writers for work queues, by running the `maintenance_tools -startprocess` command from the command prompt.

To run a writer for a work queue or a batch process from the command prompt

1. Start the BillingCenter server if it is not already running.
2. Open a command prompt.
3. Navigate to `BillingCenter/admin/bin`.
4. Run the following command:
   ```
   maintenance_tools -password password -startprocess process
   ```
   For the *process* value, specify a valid process code.

See also

- For a list of process codes for batch processes, including writers for work queues, see "List of Work Queues and Batch Processes" on page 120.
- For a description of the `maintenance_tools` command, see "Maintenance Tools Command" on page 188.

Terminating a Writer or Batch Process from the Command Prompt

You can terminate in-progress processes, including writers for work queues, by using the `maintenance_tools -terminateprocess` command. 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.

To determine if it is possible to terminate an in-progress process, refer to "List of Work Queues and Batch Processes" on page 120. The information for each individual batch process includes whether it is a single phase or multi-phase process. You can only stop those processes listed as being multi-phase.

In addition, it is not possible to use the `system_tools -terminateprocess` command option to terminate the operation of the `table_import` command.

To terminate a writer or batch process from the command prompt

1. Open a command prompt.
2. Navigate to `BillingCenter/admin/bin`.
3. Run the following command:
   ```
   maintenance_tools -password password -terminateprocess process
   ```
   For the *process* value, specify a valid process code or a process ID.

Checking Status of a Writer or Batch Process from the Command Prompt

You can check the status of processes, including writer processes for work queues, by using the `maintenance_tools -processstatus` command.

To check the status of a writer or batch process from the command prompt

1. Open a command prompt.
2. Navigate to `BillingCenter/admin/bin`.
3. Run the following command:

   ```
   maintenance_tools -password password -processstatus process
   ```

   For the *process* value, specify a valid process code or a process ID.

For work queues, this command returns the status of the writer process. It does not check whether there are remaining work items. It is possible for the process status to report as completed because the writer has completed adding items to the work queue, yet there are remaining unprocessed work items.

See also

- For a list of batch process codes, including work queue writer processes, see "List of Work Queues and Batch Processes" on page 120.

Scheduling Work Queue Writers and Batch Processes

The BillingCenter scheduler launches many batch processes, including writer processes for work queues, according to a schedule defined in `scheduler-config.xml`. Access this file in Guidewire Studio at **configuration** → **config** → **scheduler**. The `scheduler-config.xml` file contains entries in the following format:

```
<ProcessSchedule process="process_code" env="environment">
  <CronSchedule schedule_attributes/>
</ProcessSchedule>
```

The *process_code* is the process to run. The *environment* is an optional attribute that specifies the environment in which the schedule definition for the process applies. The *schedule_attributes* is a valid schedule specification. See "Defining a Schedule Specification" on page 114.

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 BillingCenter skips the overlapping process. If the `scheduler-config.xml` file does not list a process, then the process does not run.

Generally, schedule the amount of time between batch process runs in hours as opposed to minutes. This is because some batch processes require a lot of resources on a server. Schedule such processes to wake infrequently or at times that the server is less taxed, such as late at night or very early in the morning.

The BillingCenter scheduler uses the application server time for reference.

This topic includes:

Determining if a Batch Process Can Be Scheduled

Many batch processes, including work queue writers, can be scheduled. However, not all batch processes can be scheduled.

To determine if a batch process can be scheduled

1. Log in to BillingCenter as a user with `internaltools` and `toolsBatchProcessview` permissions.
2. Press `ALT+SHIFT+T` to access the **Server Tools**.
3. Select **Batch Process Info** if not already selected.
4. Select **Schedulable** from the drop-down. BillingCenter displays only those batch processes, including work queue writers, that can be scheduled in `scheduler-config.xml`.

Defining a Schedule Specification

The `CronSchedule` element describes when the process is run. It contains *`schedule_attributes`* that specify the exact timing, such as one time an hour or every night. The *`schedule_attributes`* is a combination of one or more of the following attributes:

Attribute	Standard Values	Default	Example
`seconds`	`0-59`	`0`	`seconds="0"`
`minutes`	`0-59`	`0`	`minutes="15"`
`hours`	`0-23`	`*`	`hours="12"`
`dayofmonth`	`1-31`	`*`	`dayofmonth="1"`
`month`	`1-12 or JAN-DEC`	`*`	`month="2"`
`dayofweek`	`1-7 or SUN-SAT`	`?`	`dayofweek="1"`

Along with the standard values listed, there are some special characters that give you more flexible options.

Character	Description
*	All values. For example, `minutes="*"` means run the process every minute.
?	Used to mean no specific value. Used only for `dayofmonth` and `dayofweek` attributes. See the examples for clarification.
-	Specifies ranges. For example, `hour="6-8"` specifies the hours 6, 7 and 8.
,	Separates additional values. For example, `dayofweek="MON,WED,FRI"` means every Monday, Wednesday, and Friday.
/	Specifies increments. For example, `minutes="0/15"` means start at minute 0 and run every 15 minutes.
L	The last day. Used only for `dayofmonth` and `dayofweek` attributes. See the examples for clarification.
W	Used for `dayofmonth` to specify the nearest weekday. For example, if you specify `1W` for `dayofmonth`, and that day is a Saturday, the trigger will fire on Monday the 3rd. You can combine this with `L` to schedule a process for the last weekday of the month by specifying `dayofmonth="LW"`.
#	used to specify "the nth" day of the week within a month. For example, a `dayofweek` value of 4#2 means "the second Wednesday of the month" (day `4` = Wednesday and `#2` = the second one in the month).

These represent only some of the values that you can use for setting a schedule. The BillingCenter scheduler is based on the open source Quartz Enterprise Job Scheduler, and therefore uses the same specification for schedule attributes that Quartz uses. To determine the exact Quartz version, check the filename of the Quartz JAR file in `BillingCenter/admin/lib`.

The following examples show some common ways to use the `CronSchedule` element. For additional examples, refer to the Quartz documentation. See `http://quartz-scheduler.org/documentation/quartz-2.1.x/tutorials/crontrigger` for more details and examples.

Example	Description
`<CronSchedule hours="10" />`	Run every day at 10 a.m.
`<CronSchedule hours="0" />`	Run every night at midnight.
`<CronSchedule minutes="15,45" />`	Run at 15 and 45 minutes after every hour.
`<CronSchedule minutes="0/5" />`	Run every five minutes.
`<CronSchedule hours="0" dayofmonth="1" />`	Run at midnight on the first day of the month.
`<CronSchedule hours="12" dayofweek="MON-FRI" dayofmonth="?" />`	Run at noon every weekday (without regard to the day of the month).
`<CronSchedule hours="22" dayofmonth="L" />`	Run at 10 p.m. on the last day of every month.
`<CronSchedule hours="22" dayofmonth="L-2" />`	Run at 10 p.m. on the second-to-last day of every month.
`<CronSchedule minutes="3" hours="8-18/2" dayofweek="1-5" dayofmonth="?"/>`	Run 3 minutes after every other hour, 8:03 a.m. to 6:03 p.m., Monday through Friday.
`<CronSchedule minutes="*/15" hours="0-8,18-23"/>`	Run every 15 minutes after the hour, 12:15 a.m. to 8:45 a.m. and 6:15 p.m. to 11:45 p.m.
`<CronSchedule hours="0" dayofmonth="6L" />`	Run at midnight on the last Friday of the month.
`<CronSchedule hours="4" dayofmonth="4#2" />`	Run at 4 a.m. on the second Wednesday of the month.

Determining the Current Schedule

To determine the current schedule of batch processes, you can either inspect the `scheduler-config.xml` file, or you can use the **Batch Process Info** page in BillingCenter.

To determine the schedule in BillingCenter

1. Log in to BillingCenter with an administrative account.
2. Press `ALT` + `SHIFT` + `T` to navigate to the **Server Tools** tab.
3. Click **Batch Process Info**.
4. Click the **Next Scheduled Run** column header to sort processes by schedule. If a process is not scheduled, the **Next Scheduled Run** field is blank.

Scheduling Batch Processes Sequentially to Avoid Problems

Guidewire batch processes run best if you schedule them to run sequentially so that only one batch runs at a time. If batch processes run concurrently, throughput performance can degrade significantly and a high rate of concurrent data change exceptions can occur.

For example, two separate batch processes each run on average for half an hour. If you run the two sequentially, the total time from start to finish is about an hour. You might decide to shorten the processing window by running the two batch processes concurrently. However, running the two concurrently can actually lengthen the processing window rather than shortening it.

Batch processes that run concurrently share a common cache. The cache demands of each process end up flushing the cache more frequently, so fewer cache hits occur for each process. That increases the amount of physical reads from the relational database, thus degrading performance. In addition, concurrent data change exceptions can occur when each batch process attempts to update the same cached entity instances. This requires one or the other batch process to retry an item, leading to further performance degradation.

You can use the internal BillingCenter scheduler to schedule batch processes far enough apart that they do not overlap. Alternatively, you can use your own scheduler to ensure that one batch process finishes before the next process begins.

Using the BillingCenter Scheduler to Ensure Batch Processes Do not Overlap

The internal BillingCenter scheduler does not let you specify that one batch process must finish before another one begins. The BillingCenter scheduler is purely a time-based scheduler. If you use the BillingCenter scheduler, schedule the processes in your chain of nightly batch processes far enough apart so they are unlikely to overlap.

Using Your Own Scheduler to Ensure Batch Processes Do Not Overlap

Your organization may have its own scheduler for starting batch processes. If so, you can verify that the work of one batch process is complete before the next process in your chain of nightly batch processes begins. Use the `maintenance_tools` command prompt tool or the `MaintenanceToolsAPI` web service to check the status of BillingCenter batch processes.

See also

- “Disabling the BillingCenter Scheduler” on page 116
- “Maintenance Tools Command” on page 188
- “Running Batch Processes Using Web Services” on page 128 in the *Integration Guide*

Exclude Certain Batch Processes from Running During Your Nightly Batch Window

You may want to schedule some BillingCenter batch processes to run periodically throughout the business day. For example, the default configuration of BillingCenter schedules the `ActivityEsc` batch process to run every 30 minutes. Exclude running such batch processes periodically during your nightly batch processing window. Instead, wait until the end of the batch window to run them, For example, schedule the `ActivityEsc` to run every 30 minutes except during your nightly batch window. Alternatively, run such batch processes at prescribed places in your chain of nightly batch process.

Scheduling Batch Processes for Specific Environments

You can define a schedule for each batch process for different environments. To specify an environment for the process schedule, include the `env` attribute on the `ProcessSchedule` element.

```
<ProcessSchedule process="process_code" env="environment">
  <CronSchedule schedule_attributes/>
</ProcessSchedule>
```

As a consequence, you can now have different results for batch processing based on environment.

Disabling the BillingCenter Scheduler

You can choose to use your own mechanism for running BillingCenter system processes. For example, you can use the BillingCenter API or command prompt utilities to run the processes, and you can use your own scheduling application to trigger their execution. If you do this, you might choose to disable the internal BillingCenter scheduler. To disable the internal scheduler, set the `SchedulerEnabled` configuration parameter to `false`:

```
<param name="SchedulerEnabled" value="false" />
```

Configuring Work Queues

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 the `work-queue.xml` file. For custom work queues, you must modify `work-queue.xml` to enable your work queue to operate.

Each work queue has its own configuration structure in `work-queue.xml`.

```
<work-queue workQueueClass="string" progressinterval="decimal">
  <worker instances="integer" throttleinterval="decimal" env="string" server="string"/>
  ...
  <worker instances="integer" throttleinterval="decimal" env="string" server="string"/>
</work-queue>
```

The `<work-queue>` element has attributes to configure a work queue generally. The `<worker>` subelement has attributes to configure worker threads on specific servers. You can declare as many worker instances as you want for a work queue by specifying on which servers the workers run.

You can locate `work-queue.xml` in Guidewire Studio at **configuration** → **config** → **workqueue**. After you edit `work-queue.xml`, you must rebuild and redeploy BillingCenter.

See also

- "Manipulating Work Queues Using Web Services" on page 128 in the *Integration Guide*.

General Work Queue Configuration

The `<work-queue>` element has attributes for configuring the general characteristics of a work queue.

Attribute	Description
`workQueueClass`	**Required.** The `workQueueClass` must be one of the Guidewire-provided work queue classes listed in the base version of `work-queue.xml` or a custom work queue class derived from the Gosu class `WorkQueueBase`. You cannot configure Guidewire-provided batch processes or custom batch processes derived from the Gosu class `BatchProcessBase`.
`progressinterval`	**Required.** The `progressinterval` value is the amount of time, in milliseconds, that BillingCenter allots for a worker to process `batchsize` work items. If the time a worker has held a batch of items exceeds the `progressinterval`, then BillingCenter considers the work items to be orphans. BillingCenter reassigns orphaned work items to a new worker instance. The `progressinterval` must be greater than the time to process the slowest work item, or that work item will never be completed. Also, Guidewire recommends that you set `progressinterval` greater than the processing time for an entire `batchsize` of work items. If a worker takes more time than `progressinterval` to processes its assigned work items, BillingCenter reverts the remaining work items to `available` from `checkedout`. If many worker batches take longer than `progressinterval`, the repeated checking out and reverting to available of work items can impact performance negatively.
`retryInterval`	**Optional.** How long in milliseconds to wait before retrying a work item that threw an exception. The default value is `0`, meaning BillingCenter retries processing the item immediately.
`retryLimit`	**Optional.** How many times BillingCenter retries a work item that threw an exception or became an orphan for this work queue. If you do not specify a `retryLimit` value for a work queue, BillingCenter 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.
`logRetryableCDCEsAtDebugLevel`	**Optional.** If `logRetryableCDCEsAtDebugLevel` is set to `true` for a work queue, BillingCenter logs any retryable Concurrent Data Change Exception (CDCE) at the DEBUG level. The log message includes a prepended string indicating that the error is considered to be non-fatal. Any CDCE that pushes the retry count over the `retryLimit`, or `workItemRetryLimit` if `retryLimit` is not set, is logged at the ERROR level.

Worker Thread Configuration

The <worker> subelements of the <work-queue> element has attributes for configuring workers on specific servers.

Attribute	Description
instances	**Optional.** The number of workers to create. By default, BillingCenter always creates at least one worker. Guidewire recommends an upper limit of 100 instances per server.
maxpollinterval	**Optional.** How often a worker will wake up automatically and query for work items, even if the worker receives no notification. If a worker wakes up and detects work items, it will check out those work items. If there are more work items than the batchsize, the worker will start another worker. Each worker will check out up to batchsize work items and then start another worker if there are more work items remaining until the number of instances is reached. You might need to increase maxpollinterval to prevent excessive numbers of queries for work items. The default maxpollinterval is 60,000 milliseconds.
throttleinterval	**Optional.** The delay between processing work items in milliseconds. The value controls how long the process sleeps. A value of 0 (zero) means worker threads process work items as rapidly as possible. To reduce the CPU load, set the throttleinterval to a positive value.
batchsize	**Optional.** How many work items the worker attempts to check out while searching for more work items. Larger batch sizes are more efficient, but might not result in good load distribution. The default batchsize is 10.
env	**Optional.** The environment in which this particular worker is active.
server	**Optional.** The server on which this particular worker is active.

Note: If you do not specify workers for a queue, or you do not specify the env and server attributes, BillingCenter starts a single instance on the batch server.

See also

- For information about the env and server attributes, see “Specifying Environment Properties in the <registry> Element” on page 15.

Worker Thread Management

An *executor* manages the worker threads on each server. One executor is created per work queue per server. The executor creates a single worker at server start up, regardless how many workers are configured for that server. The lone worker then periodically checks the work queue for work items on the configured frequency. If the worker finds work items, the executor creates additional worker threads until all configured workers are active or the work queue has no non-failed work items. After the work queue is exhausted, the executor shuts down all workers except one.

Performing Custom Actions After Batch Processing Completion

You can use Process Completion Monitor batch 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 batch 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 ensure that it invokes 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 completed a batch of work. The parameters of the `completed()` method are the `ProcessHistory` entity and the number of failed items. A work queue batch is considered complete if no work items remain on the queue, other than work items that failed. A batch process batch is complete if the process stopped and its process history is available.

See also

- "Process Completion Monitor Batch Processing" on page 136

By default, BillingCenter Guidewire schedules the Process Completion Monitor to run every five minutes. You can alter the schedule by modifying the `CronSchedule` element for the `ProcessCompletionMonitor` process. See "Defining a Schedule Specification" on page 114.

To create a class to implement the IBatchCompletedNotification interface

1. In Studio, expand **configuration** → **gsrc**. If you have already created a package for your plugins within **gsrc**, navigate to that package.
2. Right-click **gsrc** or your custom package, and then click **New** → **Package**.
3. Enter a package name, such as `workqueue`.
4. Right-click the new package and click **New** → **Gosu Class**.
5. Enter the name `IBatchCompletedNotification` for the gosu class.
6. Click **OK**.
7. Define your Gosu class, using the following framework as an example:

```
package myCompany.plugin.workqueue
uses gw.plugin.workqueue.IBatchCompletedNotification

class IBatchCompletedNotification implements IBatchCompletedNotification {

  construct() {
  }

  override function completed(batch : ProcessHistory, numFailed : int) {

  //do something

  return;
  }
}
```

8. Right-click the GS file for your plugin and click **Compile** to ensure your plugin compiles successfully.
9. Save your changes.

To register the custom batch notification plugin

1. In Studio, expand **configuration** → **config** → **Plugins**.
2. Right-click **registry** and click **New** → **Plugin**.
3. In the **Plugin** dialog, enter `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 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.

Troubleshooting Work Queues

Workers can encounter problems in processing that cause the worker to fail before completing items that the worker has checked out. For example, a server might die, killing its workers in the middle of processing. This can result in orphan work items. Orphans are created if a worker has an item checked out but does not complete processing the item within the allotted `progressinterval` (current time - `LastUpdateTime` > `progressinterval`). Workers treat orphans just as they do `available` items. The next worker that encounters the orphan item in the table adopts it for processing and resets the `LastUpdateTime`, `CheckedOutBy`, and `Status` fields on the orphan work item.

If a work queue is experiencing a large number of orphans, review log files to locate timeouts during processing. For example, a timeout might be caused by a worker waiting for an external server to return a value. If the log contains these type of timeouts, increase the `progressinterval` value to give workers more processing time.

Sometimes, a problem inherent in the item itself causes processing of the item to fail. For example, an exception is thrown. In such cases, the worker stops processing the item and goes on to the next. The item becomes orphaned and the next worker attempts to process it. In this way, a work queue attempts to process each item multiple times up to a limit configured for the work queue. If a work item exceeds the limit of processing attempts, BillingCenter changes the status of the work item to `failed`. Workers ignore `failed` items and no longer attempt to process them.

To remove failed work items, run the Purge Failed Work Items process. Run the Purge Failed Work Items process twice to complete the purge. The first run sets the date on the failed work items and the second run performs the actual purge.

See also

- “Purge Failed Work Items Batch Processing” on page 138

List of Work Queues and Batch Processes

BillingCenter has a number of work queues and batch processes for different kinds of batch processing supported by BillingCenter. The following topics describe the work queues and batch processes provided with BillingCenter.

See also

- “Running Work Queue Writers and Batch Processes” on page 111
- “Terminating a Writer or Batch Process from the Command Prompt” on page 112
- “Scheduling Work Queue Writers and Batch Processes” on page 113
- “List of Unused and Internal Batch Processes” on page 144
- “Batch Process Info” on page 156
- “Work Queue Info” on page 157

Account Inactivity Batch Processing

Code	`AccountInactivity`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:35 a.m.
Class	`AccountInactivityWorkQueue.java`

Account Inactivity batch processing creates notification activities for accounts that are inactive for longer than a defined number of days and do not already have an open `AcctInactiveActivity` activity.

Account Inactivity batch processing operates on accounts that:

- Do not have an open `Activity` of type `AcctInactiveActivity`.
- Do not have any `AccountHistory` records that were created within the account activity period with the `CountsAsActivity` flag set to `True`.

Account Inactivity batch processing creates a notification activity that is a subtype of `Activity` called `AcctInactiveActivity` with a status of open.

If the process detects an open `AcctInactiveActivity`, it does not create additional `AcctInactiveActivity` activities for that account. If a user changes the status to something other than open, then `AccountInactivity` creates a new `AcctInactiveActivity` for that account.

BillingCenter uses the `DaysBeforeAccountIsConsideredInactive` system parameter to construct the time window in which history records must be created for an account to be considered active. The time window is defined as:

```
today >= history event >= (today - DaysBeforeAccountIsConsideredInactive)
```

For each account that Account Inactivity batch processing finds, the batch process calls the following static Gosu method:

```
gw.activity.AccountInactivityUtil.maybeCreateAcctInactiveActivity
```

Activity Escalation Batch Processing

Code	`ActivityEsc`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Every 30 minutes
Class	`ActivityEscalationWorkQueue.java`

Activity Escalation batch processing finds activities that meet certain escalation criteria and marks the activity for escalation. The batch processing logic looks for activities that meet each of the following criteria:

- The activity has an escalation date.
- The escalation date has passed.
- The activity has not already been escalated.

If Activity Escalation batch processing 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, BillingCenter runs Activity Escalation batch processing every 30 minutes. As indicated, you can change this schedule as needed.

See also

- "Administering Activity Patterns" on page 379 in the *Application Guide*
- "Defining Activity Patterns" on page 373 in the *Configuration Guide*
- "Activity Escalation Rules" on page 37 in the *Rules Guide*

Advance Expiration Batch Processing

Code	`AdvanceExpiration`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:15 a.m.
Class	`AdvanceExpirationWorkQueue.java`

Advance Expiration batch processing checks for advance commission payments that are no longer maintained and expires them. Any remaining advance balance is then owed by the producer.

The Advance Expiration batch process query returns `AdvanceCmsnPayment` entities that meet the following criteria:

- Are not expired
- Have a `MaintainUntil` date that is before or equal to the current date and time

For each `AdvanceCmsnPayment` found, the Advance Expiration process calls the `Producer.expireAdvances()` method. This method expires the advance commission payment. This is done by removing any amount in excess of the total of all maintained advances from the producer's `CommissionsAdvance` T-account through the `ProducerAdvanceExpired` transaction.

For each `AdvanceCmsnPayment` that `AdvanceExpiration` finds, the process sets `AdvanceCmsnPayment.Expired` to `True`.

Agency Suspense Payment Batch Processing

Code	`AgencySuspensePayment`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:30 a.m.
Class	`AgencySuspensePaymentWorkQueue.java`

Agency Suspense Payment batch processing finds all active agency bill suspense payment items. A suspense item is a place to temporarily put funds that are targeted for a policy period that has not yet been created. This is necessary as agency cycle payments and promises can target policy periods that do not yet exist in BillingCenter but might already exist in the policy administration system.

For each item that the batch processing finds, the process calls the `matchAgencyBillSuspensePolicyPeriod()` method in the `EventHandler` plugin implementation to match the item with a policy period. If no match is found, the item is unchanged.

If the method finds a matching PolicyPeriod, it sets the MatchingPolicy field of the suspense item to that policy period. The process then calls the processAgencyBillSuspenseItemAfterPolicyMatch() method of the EventHandler plugin implementation. The default implementation of this method does nothing at all. However, you can provide custom logic for this method if you want to perform any special processing after a suspense item is matched with a policy period. For example, you might want to apply the funds associated with the suspense item to the policy period.

Allot Funds for Funds Tracking Batch Processing

Code	FundsAllotment
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 4:10 a.m.
Class	FundsAllotmentWorkQueue.java

Allot Funds for Funds Tracking batch processing operates on Account entity instances that have funds to allot. The process then calls the Account.allotAllFunds() method for each Account instance in the result set. The allotAllFunds() method finds source trackers and use trackers on the account that are not fully allotted. It packages the source and use trackers into two sets, which it passes as parameters to the allotFunds() method on the IFundsTracking plugin implementation. The plugin implementation performs an allotment process against each Account instance.

For more information, see “Funds Tracking Plugin” on page 198 in the *Integration Guide*.

Automatic Disbursement Batch Processing

Code	AutomaticDisbursement
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 4:00 a.m.
Class	AutomaticDisbursementWorkQueue.java

Automatic Disbursement batch processing operates on accounts with unapplied balances and that have a positive balance in their Unapplied T-account. The process either uses those funds to pay new charges, or creates or modifies a disbursement that, if approved, results in funds being returned to a customer.

For each account that qualifies for processing, the Automatic Disbursement process:

1. Distributes funds in the Unapplied T-account as follows:
   - If there is no payment distribution hold on the account, the Automatic Disbursement process goes through existing non-zero dollar payments (backwards, from most recent) and distributes them. Any payments that have undistributed amounts will draw from available funds in the account's Unapplied T-accounts to create distribution transactions. An Unapplied T-account is not allowed to go below zero.
   - If there is a remaining balance on the account (outstanding plus unbilled items) and there are funds remaining in an Unapplied T-account, a ZeroDollarDMR is created and distributed. This process attempts to distribute all remaining funds in the Unapplied T-account to any outstanding items according to the account's Distribution Limit Type, such as “Up to amount billed”.

2. Checks the amount available for disbursement on the account, which is either `Unapplied – Remaining Balance`, or `Unapplied – Billed and Due` depending on the account's `AvailableDisbAmtType`. The total amount of the existing disbursements on the account is subtracted from this value to arrive at the `additionalAmountToDisburse` for this account.
3. Updates the most recent pending disbursement instead of creating a new one, if the following criteria are met:
   - There is at least one pending disbursement.
   - The `additionalAmountToDisburse` is negative (less money available to disburse).
   - `AllowModOfManDisb` is set to `True` on the account's billing plan.
   - The most recent pending disbursement is automatic.

   The amount on the disbursement is lowered to reflect the new amount available for disbursement. However, it is not allowed to go below zero.
4. Creates a new automatic disbursement if the previous step's criteria are not met and the following criteria are met:
   - The `additionalAmountToDisburse` is greater than the billing plan's **Review Disbursement Over** amount.
   - There is no distribution hold on the `Unapplied` T-account. The process queries the Account Distribution Limit plugin for the existence of a hold. If a hold exists, then funds in the `Unapplied` T-account are not distributed. For more information, see "Account Distribution Limit Plugin" on page 187 in the *Integration Guide*.
   - There are no pending disbursements, or they amount to zero.
   - The most recent pending disbursement is not automatic.
   - The billing plan's `AllowModOfManDisb` is set to `False`.
5. Calls the `shouldAddAmountToOldestPendingDisbursement()` method of the `Disbursement` plugin implementation, if the previous step's criteria are not met and the following criteria are met:
   - The value of `additionalAmountToDisburse` is greater than the billing plan's **Review Disbursement Over** amount.
   - There is no distribution hold on the `Unapplied` T-account.

   If the method returns `False`, BillingCenter creates a new pending disbursement.

   If the method returns `True`, BillingCenter adds the value of `additionalAmountToDisburse` to the most recent (not the oldest) pending disbursement. If the pending disbursement was already approved, BillingCenter sets its `Approval` status to `Awaiting Approval` and start an approval activity for it if necessary. Otherwise, BillingCenter checks to make sure the current approving user has the authority necessary to approve the updated disbursement amount. If they do not, BillingCenter reassigns the approval activity to someone with the requisite authority.

Charge ProRata Transaction Batch Processing

Code	`ChargeProRataTx`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:20 a.m.
Class	`ChargeProRataTxnWorkQueue.java`

Charge ProRata Transaction batch processing operates on `ChargeProRataTx` entities with an `EarnDate` that has passed and a `Status` of `Planned`. The process creates a `ProRataChargeEarned` transaction for each `ChargeProRataTx` entity that it finds, using `ChargeProRataTx.Amount`. The transaction moves funds from the `Unearned` T-account for the charge to its `Earned` T-account.

BillingCenter creates ChargeProRataTx entities as it creates a Charge with a ChargePattern subtype of ProRataCharge.

Collateral Requirement Effective Batch Processing

Code	CollEffective
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 2:45 a.m.
Class	CollEffectiveWorkQueue.java

Collateral Requirement Effective batch processing operates on pending CollateralRequirement entities with an EffectiveDate on or before the current date. The process changes the Compliance status from Pending to Not Compliant and flags the collateral for processing by the collateral allocation algorithm. The algorithm then tries to allocate existing funds to ensure that all active requirements associated with a collateral requirement are satisfied.

Collateral Requirement Effective batch processing operates on CollateralRequirement entities that meet the following criteria:

- Have a Compliance of Pending
- Have an EffectiveDate that is before or on the current date

For each qualifying CollateralRequirement entity, the process updates CollateralRequirement.Compliance to Not Compliant, and sets a flag on its related Collateral entity. Upon commit, BillingCenter runs the collateral allocation algorithm, which uses available collateral, such as cash or letters of credit, to fulfill the various collateral requirements.

Collateral Requirement Expiration Batch Processing

Code	CollExpiration
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 2:50 a.m.
Class	CollExpirationWorkQueue.java

Collateral Requirement Expiration batch processing operates on CollateralRequirement entities with an ExpirationDate on or before the current date. The process changes the Compliance status to Closed and flags the collateral for processing by the collateral allocation algorithm. This algorithm then tries to allocate existing funds to ensure that all active requirements associated with a collateral requirement are satisfied.

Collateral Requirement Expiration batch processing operates on CollateralRequirement entities that meet the following criteria:

- Have a Compliance of Compliant or Not Compliant
- Have an ExpirationDate that is before or on the current date

For each qualifying CollateralRequirement entity, the process updates CollateralRequirement.Compliance to Closed and sets a flag on its related Collateral entity. Upon commit, BillingCenter runs the collateral allocate algorithm, which uses available collateral, such as cash or letters of credit, to fulfill the various collateral requirements.

Commission Payable Calculations Batch Processing

Code	CmsnPayable
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 3:50 a.m.
Class	CommissionPayableWorkQueue.java

Commission Payable Calculations batch processing makes commission payable on direct bill polices. The process looks for commissions that are eligible to be earned by a producer. For each eligible commission found, the process creates a reserve commission earned transaction. The transaction debits the policy's CommissionReserved T-account and credits the policy's CommissionPayable T-account.

In most cases, commission becomes payable as soon as the PayableCriteria on the applicable CommissionSubPlan occurs. A criterion, such as receiving a payment or billing an invoice, serves as the triggering event to make commission immediately payable. The following criteria do not automatically trigger commission earning:

- On Effective Date
- On Expiration Date
- Custom

Commission Payment Batch Processing

Code	CommissionPmt
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 2:30 a.m.
Class	CommissionPaymentWorkQueue.java

Commission Payment batch processing executes manual commission payments that are scheduled for execution on a specific date.

Commission Payment processing operates on DelayedCmsnPayment entities that:

- Have a PayOn date that is before or on the current date.
- Have a PaymentTime setting of payondate.
- Have a Paid status of False.
- Have an ApprovalStatus of Approved.

For each qualifying DelayedCmsnPayment entity, the Commission Payment process creates a ManualPayment entity and sets ManualPayment.Producer to the Producer associated with the DelayedCmsnPayment.

Each DelayedCmsnPayment entity then calls execute(ManualPayment), which creates commission payment transactions that differ per subtype of DelayedCmsnPayment:

- First, the CommissionPayment supertype execute() method is called, which sets the DelayedCmsnPayment's Paid status to True and sets the ProducerPayment field to the newly created ManualPayment entity.
- The AdvanceCmsnPayment subtype creates a ProducerAdvancePayable transaction, which debits the Producer's CommissionAdvance T-account, and credits the Producer's CommissionsPayable T-account.

- The BonusCmsnPayment subtype creates a ProdCodeBonusEarned transaction, which debits the BonusCmsnPayment.ProducerCode's CommissionExpense T-account and credits the BonusCmsnPayment.ProducerCode's CommissionsPayable T-account. It then creates a ProdCodeCmsnPayable transaction, which moves the funds just added to the ProducerCode's CommissionsPayable T-account from that account to the Producer's CommissionsPayable T-account.
- The ManualCmsnPayment subtype creates a ManualCmsnEarned transaction, which debits the ManualCmsnPayment.ProducerCode's CommissionExpense T-account and credits the ManualCmsnPayment.ProducerCode's CommissionsPayable T-account. It then creates a ProdCodeCmsnPayable transaction, which moves the funds just added to the ProducerCode's CommissionsPayable T-account from that account to the Producer's CommissionsPayable T-account.
- The StandardCmsnPayment subtype just moves the balance of the ProducerCode's CommissionsPayable T-account to the Producer's CommissionsPayable T-account using the ProdCodeCmsnPayable transaction, when the PaymentType of the StandardCmsnPayment is All Current. If the PaymentType is Other Amount, then BillingCenter attempts to move the amount specified on the StandardCmsnPayment to the Producer's CommissionsPayable T-account. If the amount requested is greater than the balance of the ProducerCode's CommissionsPayable T-account, then the balance is obtained from the ProducerCode's PolicyProducerCodes and PolicyProducerCodeCharges CommissionsPayable balances. If there are still insufficient funds, then whatever amount was found is transferred to the Producer's CommissionsPayable T-account.

Database Consistency Check Batch Processing

Code	DBConsistencyCheck
Categories	APIRunnable, Schedulable
Implementation	Work queue
Non-exclusive	Yes
Schedule	Not scheduled
Class	DBConsistencyCheckWorkQueue.java

Note: Database Consistency Check batch processing is non-exclusive, meaning that is possible to run multiple instances of the batch process against the database.

The Database Consistency Checks batch processing runs consistency checks on the BillingCenter database. Database consistency batch processing runs only on the batch server.

Do not launch this batch process using the maintenance_tools command. You cannot specify which checks to run or which tables to run the checks against with the maintenance_tools command. Instead, use the **Server Tools** → **Info Pages** → **Consistency Checks** screen to launch the checks from BillingCenter.

Or, if you want 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

- "Checking Database Consistency" on page 36 for an overview of database consistency checks
- "Consistency Checks" on page 161 for details of the Consistency Checks page in BillingCenter
- "System Tools Command" on page 191 for an explanation of command prompt options

Database Statistics Batch Processing

Code	`DBStats`
Categories	`Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`DBStatisticsWorkItemWorkQueue.java`

The Database Statistics batch processing generates database statistics about how the BillingCenter application and data model interact with the physical database. For example, database statistics store row counts in a table, how the data is distributed in a table, and much more. A database management system uses statistics to determine query plans to optimize performance.

You run Database Statistics batch processing from the command prompt only, using a `system_tools` command. See "Commands for Updating Database Statistics" on page 40 for the specific `system_tools` commands that you can use to generate and update 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 `table_import -integritycheckandload` or `zone_import` command, or if there are performance issues.

See also

- "Configuring Database Statistics" on page 39.

Deferred Upgrade Tasks Batch Processing

Code	`DeferredUpgradeTasks`
Categories	`APIRunnable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not schedulable
Class	`DeferredUpgradeBatchProcess.java`

Deferred Upgrade Tasks batch processing creates the nonessential performance indexes. BillingCenter runs Deferred Upgrade Tasks batch processing automatically after an upgrade if you set the following attribute on `<upgrade>` in `database-config.xml` to true:

```
defer-create-nonessential-indexes
```

If `DeferredUpgradeTasks` batch processing fails, manually run the batch process again during non-peak hours. To manually run the Deferred Upgrade Tasks batch processing, 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 processing, review the upgrade logs and the BillingCenter **Upgrade Info** page.

See also

- "Upgrade Info" on page 171
- "Maintenance Tools Options" on page 189

- “Deferring Creation of Nonessential Indexes” on page 155 in the *Upgrade Guide*.

Disbursement Batch Processing

Code	`Disbursement`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 4:05 a.m.
Class	`DisbursementWorkQueue.java`

Disbursement batch processing handles all types of disbursements in the system, including `AccountDisbursement`, `AgencyDisbursement`, `CollateralDisbursement`, and `SuspenseDisbursement`. The process creates outgoing disbursement payments. This process checks for approved disbursements with due dates that have passed and creates the disbursement to pay the disbursement amount to the account.

The process first queries for all disbursements that either are approved already or are automatic and awaiting approval.

The process then filters out any returned disbursements that are of type `AccountDisbursement` and associated with an account that is currently the target of a trouble ticket disbursement hold.

Then, the process passes the disbursement to the `updateDisbursementStatusAndAmount()` method of the `IEventHandler` plugin implementation, which processes the disbursement as follows:

1. The process inspects the `DueDate` field of the `Disbursement` entity. If the `DueDate` is after the current date, then the process filters out the disbursement.
2. The process inspects the `Status` of the disbursement. The disbursement is executed immediately if the `Status` is `DisbursementStatus.TC_APPROVED` or the disbursement:
   - Has `DisbursementStatus.TC_AWAITINGAPPROVAL`.
   - Is an `AccountDisbursement`.
   - Has an account associated with a `BillingPlan` that has `SendAutoDisbAwaitingApproval` set to `True`. (This is the **Send automatic disbursements awaiting approval on their due dates** field on the **Billing Plan** screen).
3. Otherwise, the disbursement is not executed.

Full Pay Discount Batch Processing

Code	`FullPayDiscount`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:00 a.m.
Class	`FullPayDiscountWorkQueue.java`

Full Pay Discount batch processing operates on `PolicyPeriod` entities that meet the following criteria:

- `EligibleForFullPayDiscount` set to `true`
- `FullPayDiscountEvaluated` set to `false`
- `FullPayDiscountUntil` date before the current date

For each qualifying PolicyPeriod, the process calls the applyFullPayDiscount() method on the implementation of the applyFullPayDiscount plugin. The default implementation of this method uses the PolicyPeriod.DiscountedPaymentThreshold property to determine whether to apply a discount and the discount amount. Modify this plugin method to fit your requirements for full-pay discounts.

Invoice Batch Processing

Code	Invoice
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 3:20 a.m.
Class	InvoiceBilledWorkQueue.java

Invoice batch processing operates on AccountInvoice entities that have met the following criteria:

- Status field equal to Planned
- EventDate plus the cutoff time is less than the current date

The process sets the Status of matching account invoice entities to Billed and executes the necessary transactions on the invoice items.

InvoicePeriodCutoff is a configuration parameter that takes a time value and is used to calculate the time of day that an invoice period ends. The process subtracts the InvoicePeriodCutoff time from the current date and time. Any unbilled invoice with a due date prior to the resulting date and time is considered for billing.

The first step in billing an AccountInvoice is to push any held InvoiceItems forward to a future AccountInvoice. The set of held InvoiceItems includes any non-separate InvoiceItems that the IInvoice.shouldHoldDirectBillInvoiceItem() plugin method determines must be held.

The next step is to determine if the entire AccountInvoice is carried forward. This is determined by calling IInvoice.shouldCarryForwardInvoice(AccountInvoice). If this plugin method returns true, then all InvoiceItems on the current AccountInvoice are added to the next planned AccountInvoice, and the current AccountInvoice.Status is marked as Carried Forward.

If the AccountInvoice meets the following conditions, then a writeoff is created for the AccountInvoice and AccountInvoice.Status is marked as Written Off. The conditions are:

- AccountInvoice.Account.BillingPlan.SuppressLowBalInvoices is set to true.
- AccountInvoice.Amount is less than AccountInvoice.Account.BillingPlan.LowBalanceThreshold.
- AccountInvoice.Account.BillingPlan.LowBalanceMethod equals Write Off.

If the AccountInvoice is neither written off nor carried forward, then invoice fees are added as necessary. Invoice fees are not added to account invoices that have been pre-paid. The amount of the invoice fee is determined by calling the IInvoice.getInvoiceFeeAmount() plugin implementation method. If the invoice fee amount is non-zero, then a Charge with the Invoice Fee ChargePattern is added to the AccountInvoice.Account.

If AccountInvoice.Account.BillingPlan.SkipInstallmentFees is false, then installment fees are added as necessary. Installment fees are only added for InvoiceItems that are linked to a PolicyPeriod. Installment fees are added for each PolicyPeriod on a given AccountInvoice. An installment fee is also added for the deposit if the PolicyPeriod payment plan's **SkipFeeForDownPayment** flag is false. The amount of the installment fee for a given PolicyPeriod is determined by calling the IInvoice.getInstallmentFeeAmount() plugin method.

If the AccountInvoice.Account.PaymentMethod is **Credit Card**, **ACH**, or **Wire**, then a PaymentRequest entity is created. The StatusDate, RequestDate, and DraftDate for the PaymentRequest are initialized to be the AccountInvoice.EventDate plus the AccountInvoice.Account.BillingPlan.RequestInterval. Initially, the PaymentRequest.Status is Created. Payment Request batch processing later updates PaymentRequest.Status.

The AccountInvoice.Status is updated from Planned to Billed.

Each InvoiceItem on the AccountInvoice executes a billed transaction. If InvoiceItem.Amount is positive, then the ChargeBilled transaction is used. This transaction moves funds from the Unbilled T-account to the Billed T-account of the InvoiceItem.Charge. If InvoiceItem.Amount is negative, then the NegativeChargeBilled transaction is used. This transaction moves funds from the Unbilled T-account of the InvoiceItem.Charge, through its Billed and Due T-accounts to the T-account owner's Unapplied T-account. In other words, the NegativeChargeBilled transaction pushes the credit represented by the negative charge into the Unapplied T-account, where that credit can then be used to pay positive charges.

After all invoicing transactions have been executed, the charge payment algorithm is called on all T-account owners related to the AccountInvoice. This call allows positive invoice items to be paid by the credit from negative invoice items.

After all invoice items have been billed and credits applied, snapshot data is generated for the AccountInvoice. This snapshot data captures the balances of the AccountInvoice at the time it was billed. The snapshot data is stored on the AccountInvoice itself, and is populated by calls to the following plugin methods on IInvoice:

- getOutstandingAmount()
- getRemainingBalance()
- getUnappliedAmount()
- getCollateralOutstandingAmount()
- getCollateralRemainingBalance()
- getCollateralUnappliedAmount()
- setPolicyPeriodInvoiceSnapshot() for each PolicyPeriod related to the AccountInvoice.

Invoice Due Batch Processing

Code	InvoiceDue
Categories	UIRunnable, Schedulable
Implementation	Work queue
Schedule	Once a day, at 3:15 a.m.
Class	InvoiceDueWorkQueue.java

Invoice Due batch processing marks appropriate invoices as past due. BillingCenter selects an invoice for processing based on its due date and an adjusted current date and time. The current date and time is adjusted by subtracting the time specified in the InvoicePeriodDueCutoff configuration parameter. The default time of InvoicePeriodDueCutoff is 12:00 a.m. All invoices with a due date prior to the adjusted time are selected for additional processing.

If a selected invoice has a Status of Billed and its due date has passed, Invoice Due batch processing changes its Status to Due. The batch processing also starts a delinquency process for any past due invoice amounts.

If a selected invoice has a Status of Carried Forward or Written Off and meets the following criteria, Invoice Due batch processing creates the transactions:

- The invoice due date has passed.
- The invoice contains items without ChargeDue transactions.

Invoice Due batch processing leaves the invoice Status unchanged.

Legacy Agency Bill Batch Processing

Code	`LegacyAgencyBill`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Worker instances	0
Schedule	Once only, if needed
Class	`LegacyAGBLWorkQueue.java`

IMPORTANT Run Legacy Agency Bill batch processing once only, after loading staging tables during a conversion project. Run this process with exactly one worker as it is not safe to run with multiple workers. Disable Legacy Agency Bill batch processing in a production environment.

Legacy Agency Bill batch processing starts workflows after successful completion of a staging table load. The process determines those agency cycle processes for which the agency bill plan is Legacy Agency Bill Plan. For those agency cycle process, the process sets the `AgencyBillWorkflow` attribute of the agency cycle process.

By default, Guidewire disables Legacy Agency Bill batch processing in the base configuration by setting the number of worker instances in `work-queue.xml` to 0. You must set the number of worker instances in this file to 1 to be able to run this process.

Disable this process after batch processing completes by resetting the number of worker instances back to 0.

See also

- “Configuring Work Queues” on page 116

Legacy Collateral Batch Processing

Code	`LegacyCollateral`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Worker instances	0
Schedule	Once only, if needed
Class	`LegacyCollateralWorkQueue.java`

IMPORTANT Run Legacy Collateral batch processing once only, after loading data from staging tables during a conversion process. Run this batch processing with exactly one worker as it is not safe to run with multiple workers. Disable Legacy Collateral batch processing in a production environment.

Legacy Collateral batch processing must be run after `Collateral` or `LetterOfCredit` entities have been imported into the system through the staging tables.

The process runs the collateral allocation method `CollateralImpl.touchAndMarkComputedAmountsDirty()` on all `Collateral` entities in the system. This accomplishes updating the denormalization fields on the `Collateral` entities, which are otherwise too complicated to update in a loader callback.

By default, Guidewire disables Legacy Collateral batch processing in the base configuration by setting the number of worker instances in `work-queue.xml` to 0. You must set the number of worker instances in this file to 1 to be able to run this process.

Disable this process after batch processing completes by resetting the number of worker instances back to 0.

See also

- "Configuring Work Queues" on page 116

Legacy Delinquency Batch Processing

Code	`LegacyDelinquency`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Worker instances	0
Schedule	Once only, if needed
Class	`LegacyDelinquencyWorkQueue.java`

IMPORTANT Run Legacy Delinquency batch processing once only, after loading data from staging tables during a conversion process. Run this batch processing with exactly one worker as it is not safe to run with multiple workers. Disable Legacy Delinquency batch processing in a production environment.

IMPORTANT Before you run this process, create a delinquency plan in BillingCenter based closely on the logic used in the legacy system. This logic must correspond to the events in the workflow section of the delinquency plan.

Legacy Delinquency batch processing operates on delinquency processes that meet the following criteria:

- Has an associated delinquency plan with the name Legacy Delinquency Plan
- Has a status of `Open`
- Has no associated workflow

After the load, this process starts a workflow for each active delinquency process. Delinquency processes with an `Open` status are active. Delinquency processes with a `Closed` status are past. They are for historical purposes, so their events have happened already. You can see these past events from the **Delinquency** screen for the account.

By default, Guidewire disables Legacy Delinquency batch processing in the base configuration by setting the number of worker instances in `work-queue.xml` to 0. You must set the number of worker instances in this file to 1 to be able to run this process.

Disable this process after batch processing completes by resetting the number of worker instances back to 0.

See also

- "Configuring Work Queues" on page 116

Legacy Letter of Credit Batch Processing

Do not use. Run Legacy Collateral batch processing instead.

Letter Of Credit Batch Processing

Code	`LetterOfCredit`
Categories	`UIRunnable`, `Schedulable`

Implementation	Work queue
Schedule	Once a day, at 2:40 a.m.
Class	`LetterOfCreditWorkQueue.java`

Letter Of Credit batch processing operates on `LettersOfCredit` entities with a status of `Current` using the `LetterOfCreditStatus.TC_CURRENT.get()` method and an `ExpirationDate` that is before the current date. For qualifying `LettersOfCredit` entities, the process changes their status to `Expired`. This causes the collateral allocation algorithm to run.

The collateral allocation algorithm then uses available collateral, such as cash or letters of credit, to fulfill the various collateral requirements.

New Payment Batch Processing

Code	`NewPayment`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:10 a.m.
Class	`NewPaymentWorkQueue.java`

New Payment batch processing allocates automatic payments on an account to the corresponding account or policy charges. The process also allocates payments that have been receipted manually using the Enter Multiple Payments screen. This process distributes available money according to the account distribution settings.

The New Payment process distributes the money as follows:

- Searches for direct bill money that has been received (`DirectBillMoneyRcvd`) but not distributed to a payment and not reversed
- Calls the `distribute` method on each `DirectBillMoneyRcvd` entity found

Payment Request Batch Processing

Code	`PaymentRequest`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:25 a.m.
Class	`PaymentRequestWorkQueue.java`

Payment Request batch processing selects payment requests that have their `StatusDate` field falling on or before the current date and the `Status` field that is not closed or canceled. Selected requests have their `StatusDate` and `Status` fields advanced.

Best practice recommends that whenever `StatusDate` is changed, the `RequestDate` field be assigned the equivalent setting. The `RequestDate` field is shown in some parts of the user interface.

Phone Number Normalizer Batch Processing

Code	`PhoneNumberNormalizer`
Categories	`UIRunnable`
Implementation	Work queue
Worker instances	0
Schedule	Once only, after an upgrade to 8.0.0+
Class	`CompactPhoneNormalizerWorkQueue.java`

IMPORTANT Run Phone Number Normalizer batch processing once only, after upgrading from earlier application versions to 8.0.0+. Disable Phone Number Normalizer batch processing in a production environment.

Phone Number Normalizer batch processing calls the registered plugin that implements the `IPhoneNumberNormalizer` interface. Use Phone Number batch processing to normalize phone numbers after upgrading from earlier versions of BillingCenter to 8.0.0+.

By default, Guidewire disables Phone Number Normalizer batch processing in the base configuration by setting the number of worker instances in `work-queue.xml` to 0. You must set the number of worker instances in this file to 1 or more to be able to run this process.

Guidewire recommends that you use a substantial number of workers with Phone Number Normalizer batch processing. 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 will vary according to the available hardware and the volume of the data involved. It is also possible to allocate workers to several different application servers rather then simply increasing the number of workers on a single server.

Disable this work queue after the process has normalized all old phone numbers by resetting the number of worker instances back to 0. You never need run Phone Number Normalizer batch processing more than once, after an upgrade to 8.0.0+.

See also

- “Configuring Work Queues” on page 116
- See “Upgrading Phone Numbers” on page 185 in the *Upgrade Guide* for more information.

Policy Closure Batch Processing

Code	`PolicyClosure`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 4:15 a.m.
Class	`PolicyClosureWorkQueue.java`

Policy Closure batch processing on policies that meet the following criteria for the most recent policy period:

- `PolicyPeriod.ClosureStatus` is `open`. If the `PolicyPeriod.ClosureStatus` is `openlocked`, the policy cannot be closed.
- `PolicyPeriod.PolicyInfo.ExpirationDate` is in the past or `PolicyPeriod.CancelStatus` is `Canceled`.

The process then closes those qualifying policies that meet the following criteria:

- The policy `ExpirationDate` is in the past or the policy has been flat canceled. Flat canceled means that the policy is cancelled on the `PolicyPeriod EffectiveDate`.
- `PolicyPeriod.RemainingBalance` is zero.
- All pro rata charges are earned.

You can also define custom criteria to determine when a policy can be closed in `gw.plugin.eventhandler.impl.policyClosureConditionNotMet()`.

Premium Reporting Report Due Batch Processing

Code	`PremiumReportReportDue`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:55 a.m.
Class	`PremiumReportDDWorkQueue.java`

Premium Reporting Report Due batch processing checks to see whether a `PremiumReportDueDate` billing instruction became past due without receipt of a matching `PremiumReportBI` for that policy. If this has occurred, the process calls the `shouldStartFailureToReportDelinquency()` method in the `PremiumReport` plugin implementation to determine whether to start a delinquency.

In the base configuration, the method returns `true` if a `PremiumReportBI` has not been received by the due date. If the method does return `true`, BillingCenter starts a `FailureToReport` delinquency on the `PremiumReportDueDate.PolicyPeriod`.

Process Completion Monitor Batch Processing

Code	`ProcessCompletionMonitor`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Every 5 minutes
Class	`ProcessCompletionMonitor.java`

Process Completion Monitor batch 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 ensure that it invokes 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 completed a batch of work.

See also

- “Performing Custom Actions After Batch Processing Completion” on page 118.

Process History Purge Batch Processing

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`

Process History Purge batch processing 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:** BillingCenter also uses configuration parameter `BatchProcessHistoryPurgeDaysOld` to determine how many days to retain process history records, which the separate Work Item Set Purge Batch Processing process removes.

Producer Payment Batch Processing

Code	`ProducerPayment`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:55 a.m.
Class	`ProducerPaymentWorkQueue.java`

Producer Payment batch processing operates on producers with a `Commission Day of Month` that is on or before today. For each producer that meets this criterion, the process does the following:

- It examines all `Commissions Payable` T-accounts to find all policies for which the producer has at least one role.
- It then moves the amounts from each policy's `Commissions Payable` T-account to the producer's `Commissions Payable` T-account.
- It then moves the total amount from `Producer: Commissions Payable` to `Producer: Cash`. The `Producer: Cash` T-account reflects the amount the producer will be paid. The actual payment of the producer (writing a check or transferring funds) requires integration.

The Producer Payment process generates a producer statement if the producer payment amount is greater than zero. The statement lists the commissions payable to the producer.

Purge Cluster Members Batch Processing

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`

Purge Cluster Members batch processing 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 Processing

Code	`PurgeFailedWorkItems`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Every Saturday, at 1:00 a.m.
Class	`PurgeFailedWorkItems.gs`

Purge Failed Work Items batch processing purges failed work items from all work queues. This process uses Gosu class `PurgeFailedWorkItems` to determine which work items to delete.

Purge Old Transaction IDs Batch Processing

Code	`PurgeTransactionIDs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not scheduled
Class	`PurgeTransactionIds.gs`

Purge Old Transaction IDs batch processing purges `TransactionId` entities. This process uses Gosu class `PurgeTransactionIds` to read the value of the `TransactionIdPurgeDaysOld` parameter in `config.xml`. The process then purges `TransactionId` entities that have a `CreationDate` value prior to the current date minus the value of the `TransactionIdPurgeDaysOld` parameter.

Purge Profiler Data Batch Processing

Code	`PurgeProfilerData`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	Yes (multi-phase process)
Schedule	Not scheduled
Class	`ProfilerDataPurgeBatchProcess.java`

Purge Profiler Data batch processing 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 processing removes it.

Purge Workflow Batch Processing

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`

Purge Workflow batch processing 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 Processing

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`

Purge Workflow Logs batch processing 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.

Release Charge Holds Batch Processing

Code	`ReleaseChargeHolds`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:10 a.m.
Class	`ReleaseChargeHoldWorkQueue.java`

Release Charge Holds batch processing operates on charge holds with release dates that have passed and releases them. Each charge has a `HoldReleaseDate` that marks the date when the charge is released and available for invoicing. For charge holds that have expired, the process sets the `Hold` field to `ChargeHoldStatus.TC_None`.

Release Trouble Ticket Hold Types Batch Processing

Code	`ReleaseTtktHoldTypes`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:05 a.m.
Class	`ReleaseTtktHoldTypesWorkQueue.java`

Release Trouble Ticket Hold Types batch processing operates on `HoldTypeEntry` entities on trouble tickets with release dates that have passed and releases those holds.

Statement Billed Batch Processing

Code	`StatementBilled`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:35 a.m.
Class	`StatementBilledWorkQueue.java`

Statement Billed batch processing changes the agency bill `StatementInvoice.Status` to `Billed` for all `StatementInvoices` the meet the following criteria:

- `StatementInvoice.Planned` equals `true`
- `StatementInvoice.EventDate` is equal to or before the date that the batch process runs

For each qualifying `StatementInvoice`, the process calls the `makeBilled()` method to:

- Push any invoice items that are the target of a hold to a future invoice.
- Add any installment fees for the `StatementInvoice`.
- Set a line number for each invoice item on the statement: starting at `1`, in increments of `1`, for each invoice item.
- Create charge invoicing transaction entities, such as `ChargeBilled` or `NegativeChargeBilled`, for each invoice item.
- Recalculate the following denormalized fields for the `StatementInvoice`: `Amount`, `AmountDue`, `NetAmount`, and `NetAmountPaid`.
- Start the agency bill workflow for the `StatementInvoice`.

Statement Due Batch Processing

Code	`StatementDue`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:40 a.m.
Class	`StatementBilledWorkQueue.java`

Statement Due batch processing changes the status of agency bill `StatementInvoices` to `Due` for all `StatementInvoice` entities in which:

- `StatementInvoice.PaymentDueDate` is before or on the date that the Statement Due batch process runs.

- `StatementInvoice.Billed` is `true`.

For qualifying `StatementInvoice` entities, the Statement Due batch process calls the `makeDue()` method to:

- Push any invoice items that are the target of a hold to a future invoice.
- Create `ChargeDue` transactions for each invoice item on the `StatementInvoice`.
- Recalculate the following denormalized the fields for the `StatementInvoice`: `Amount`, `AmountDue`, `NetAmount`, and `NetAmountPaid`.

Suspense Payment Batch Processing

Code	`SuspensePayment`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:05 a.m.
Class	`SuspensePaymentWorkQueue.java`

Suspense Payment batch processing searches for the payer of specific payments stored in `Unapplied` T-accounts. Specifically, payments received by BillingCenter always specify the payer. The payer can be either an account, a producer, or a policy period that is associated with an account or producer.

It is possible for BillingCenter to receive payments before it has been informed about the payer. In such cases, the account, producer, or policy period does not yet exist in BillingCenter, but will presumably be created in the near future.

When a payment is received and the payer is unknown to BillingCenter, the payment is placed in an `Unapplied` T-account of type `SuspensePayment`. Each payment with an unknown payer is placed in its own, separate `Unapplied` T-account. For example, if ten payments with unknown payers arrive then BillingCenter creates ten `Unapplied SuspensePayment` T-accounts.

Thus, Suspense Payment batch processing searches for the payer of each payment stored in these `Unapplied` T-accounts, checking to see if the account or producer has been created yet. If the payer is found to exist in BillingCenter, the payment is moved from the `Unapplied SuspensePayment` T-account to the appropriate `Unapplied` T-account of the payer.

All operations performed by the Suspense Payment process are implemented in the implementation of the configurable Suspense Payment plugin. Each suspense payment handled by the process is passed to the plugin implementation for processing. By modifying the implementation of the Suspense Payment plugin, you can configure the functionality of Suspense Payment batch processing to perform the desired operations.

See also

- “Suspense Payment Plugin” on page 197 in the *Integration Guide*.

Trouble Ticket Escalation Batch Processing

Code	`TroubleTicketEsc`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2:25 a.m.
Class	`TroubleTicketEscalationWorkQueue.java`

Trouble Ticket Escalation batch processing escalates trouble tickets that have a ticket status (`TroubleTicket.TicketStatus`) of `Open`. This process operates on unescalated trouble tickets with escalation dates that are equal to the current date or earlier and flags them as escalated.

Upgrade Billing Instruction Payment Plan Batch Processing

Code	`UpgradeBillingInstructionPaymentPlan`
Categories	`UIRunnable, Schedulable`
Implementation	Work queue
Worker instances	0
Schedule	Once only, if needed
Class	`UpgradeBillingInstructionPaymentPlanWorkQueue.java`

IMPORTANT Run Upgrade Billing Instruction Payment Plan batch processing once only, and only if you are upgrading from BillingCenter 3.0.2 or earlier to BillingCenter 8.0.0+. It is possible to run this batch processing with multiple workers. Disable Upgrade Billing Instruction Payment Plan batch processing in a production environment.

Upgrade Billing Instruction Payment Plan batch processing initializes the payment plan (`PaymentPlan`) that is on `BillingInstruction` objects that inherit from `PlcyBillingInstruction` objects.

By default, Guidewire disables Upgrade Billing Instruction Payment Plan batch processing in the base configuration by setting the number of worker instances in `work-queue.xml` to 0. You must set the number of worker instances in this file to 1 or more to be able to run this process.

Disable this process after batch processing completes by resetting the number of worker instances back to 0.

See also

- “Configuring Work Queues” on page 116

User Exception Batch Processing

Code	`UserException`
Categories	`UIRunnable, Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`UserExceptionWorkQueue.java`

User Exception batch processing runs the user exception rule set on all users in the system.

Note: BillingCenter does not provide User Exception sample rules in the base configuration.

Workflow Writer Batch Processing

Code	`Workflow`
Categories	`UIRunnable, Schedulable`
Implementation	Work queue

Schedule	Once a day, at 3:45 a.m.
Class	`WorkflowDistributedWorkQueue.java`

Workflow Writer batch processing runs workflow worker threads.

See also

- "Guidewire Workflow" on page 341 in the *Configuration Guide*

Work Item Set Purge Batch Processing

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`

Work Item Set Purge batch processing 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 Processing process removes.

Work Queue Instrumentation Purge Batch Processing

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`

Work Queue Instrumentation Purge batch processing 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

- "Work Queue Info" on page 157.

Write-off Staging Batch Processing

Code	`WriteoffStaging`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Worker instances	0
Schedule	Once only, if needed
Class	`WriteoffStagingWorkQueue.java`

IMPORTANT Run Write-off Staging batch processing once only, after loading staging tables during a conversion project. Run this process with exactly one worker as it is not safe to run with multiple workers. Disable Write-off Staging batch processing in a production environment.

Write-off Staging batch processing is used only after the staging table loading process completes. The process distributes any undistributed write-offs and reversals that result from the staging table load process. BillingCenter generates the appropriate T-accounts and transactions automatically during the loading process.

By default, Guidewire disables Write-off Staging batch processing in the base configuration by setting the number of worker instances in `work-queue.xml` to 0. You must set the number of worker instances in this file to 1 to be able to run this process.

Disable this process after batch processing completes by resetting the number of worker instances back to 0.

See also

- “Configuring Work Queues” on page 116

List of Unused and Internal Batch Processes

Some processes included with BillingCenter are used by BillingCenter internally or are not used at all. You cannot run these processes from BillingCenter, `maintenance_tools`, or an API.

Unused Processes

The Batch Process Type typelist (`BatchProcessType.tti`) includes a few Guidewire platform processes that are currently not used by BillingCenter.You can ignore these processes.

The unused processes include the following:

- Archiving Item Writer
- Bulk Purge
- ContactAutoSync
- Data Distribution
- Geocode Writer
- Group Exception
- Legacy Letter of Credit
- Populate Search Columns
- Staging Table Delete Excluded Rows
- Staging Table Encryption
- Staging Table Integrity Check

- Staging Table Load
- Staging Table Populate Exclusion Table
- Staging Table Statistics
- Stat Report Writer
- User Exception

Internal Processes

BillingCenter uses some Guidewire platform processes internally. These processes are run by BillingCenter only to generate database performance reports. You cannot run these processes separately. They are:

- Microsoft DMV Report
- Oracle AWR Report

chapter 10

Configuring Guidewire Document Assistant

BillingCenter provides the Document Assistant Java applet to enable client-side creating, viewing, editing, and uploading of files. Document Assistant can be disabled in the server configuration, in which case all such operations are handled directly by the browser.

Document Assistant supports automatic creation of new documents from custom document templates. Document Assistant opens and displays documents through Windows rather than returning the documents directly to the browser. Document Assistant directly opens documents that have an allowed file type. See “Document Assistant Supported File Types” on page 151.

For more information about document management, see “Document Management” on page 217 in the *Integration Guide*.

This topic includes:

See also

- “Localizing Document Assistant Messages” on page 45 in the *Globalization Guide*

Enabling Guidewire Document Assistant

Guidewire Document Assistant is disabled by default. The following procedure enables the Guidewire Document Assistant.

To enable Document Assistant

1. Enable Document Assistant from the `config.xml` file by setting:
   ```
   <param name="AllowDocumentAssistant" value="true"/>
   ```
2. Configure BillingCenter to display document **Edit** and **Upload** buttons by setting:
   ```
   <param name="DisplayDocumentEditUploadButtons" value="true"/>
   ```
3. Save `config.xml`.

Support for Document Management Systems

BillingCenter provides a **Documents** section for certain entity types such as an `Account`. The **Documents** section lists a set of document references that BillingCenter assumes your company stores externally, perhaps in a document management system. Guidewire provides a reference implementation for document management. This implementation assumes that documents reside on the file system. You can view this reference implementation in `BillingCenter/java-api/examples/src/examples/plugins/document`. If you do not see the `java-api` directory, run the following command from the BillingCenter `bin` directory:

```
gwbc regen-java-api
```

You can create a custom integration with a document management system. For details, see “Document Management” on page 217 in the *Integration Guide*.

Client Configuration Requirements

The Document Assistant is a signed Java Web Start (JWS) applet requiring Java Runtime Environment 7 (JRE 7) or later on the client machine. The applet is deployed in the browser by the Java Next-Generation Plugin using the Java Network Launch Protocol (JNLP).

The Document Assistant applet is inserted into the browser Document Object Model (DOM) and loaded on demand for the first operation requiring the Document Assistant. The applet and its resources are signed with the Guidewire code signing certificate, and the user is required to accept this certificate at least once. If the user elects to install the Guidewire certificate permanently in their JRE key store, they will never be prompted for authorization again. The resources are signed with a timestamp signature from a universally-recognized certificate authority. Therefore, the applet and resources will still be considered valid even after the certificate originally used to sign them has expired.

The user web client must have Java installed. For the list of supported browsers, Java versions, and operating systems, see the *Guidewire Platform Support Matrix*. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at `https://guidewire.custhelp.com/app/resources/products/platform`.

The Java Console can be used to examine diagnostic messages from the applet. Tracing and logging can be enabled through the console or with the control panel. The Java Control Panel can be used to display the currently installed version of the Document Assistant. It may also be used to examine the details of the Guidewire code signing certificate. Other messages can be seen in the browser's error console.

Document Assistant supports client-side JScript required for some templates, such as the MailMerge template. The MailMerge template is a Guidewire-supplied reference implementation. To support client-side JScript, Document Assistant creates a temporary file containing the required JScript and then runs it using Windows Script Host. To support this mechanism, the `.js` extension on the user's environment must map to Windows Script Host (`Windows\System32\wscript.exe`).

The Document Assistant is a JWS applet that uses LiveConnect (a JavaScript to Java bridge) calls to interact with the main application in the browser. The JRE security settings must allow both the applet itself to run and the LiveConnect calls to and from the applet. Document Assistant is code signed by Guidewire to allow these permissions. However, if running JRE 7u55+ or JRE 8u5+, the JRE prompts you at least once to confirm JavaScript access to the applet. To remove the need for this confirmation, it is possible to implement a deployment rule set or exception site list.

Guidewire recommends that end users stay current with Java security updates for their client JRE. Otherwise it is possible for the browser to block Document Assistant from running. Techniques described in this topic make it possible to run Document Assistant in scenarios in which it might otherwise be blocked. Deployment rule sets are intended for site administrators. Exception site lists and security sliders on the Java Control Panel are intended for end users, though it is possible for administrators to control those as well. If using exception site lists or security sliders, the browser will still report security warnings if the client java version is below the security baseline, such as:

- Java version is out of date
- Potentially unsafe components
- Warnings about the application running with unrestricted access
- Warnings about the web site referencing JavaScript code that is requesting access and control of a Java application on the web page

Creating a Deployment Rule Set

If end users are using Java 7 update 40 or above, the site administrator can create a deployment rule set to give users permission to run Guidewire Document Assistant. Without a deployment rule set, a desktop user can receive security warnings or the Guidewire Document Assistant could be blocked, depending on the Java Control Panel security level. Deployment rule sets were introduced in Java version 7 update 40.

WARNING Using a deployment rule set opens a potential security hole. The deployment rule allows any applet from the BillingCenter URL to run, signed or not, rather than just the signed Document Assistant applet. Deployment rule sets bypass all security checks and are chained from first to last until there is a match. So take extreme caution in ensuring that you do not give permissions to more applets than intended. You cannot use a certificate hash-based rule for the applet because a certificate hash-based rule only grants permission to the applet itself. It does not grant permission to the LiveConnect calls required by the applet.

To create a deployment rule set

1. Create a file called `ruleset.xml` with the following content:

```
<ruleset version="1.0+">

  <rule>
    <id location="URL for BillingCenter"/>
    <action permission="run" />
  </rule>

  <!-- Because this is both blank and shown last, it will be the default policy. -->
  <rule>
    <id />
    <action permission="default"/>
  </rule>

</ruleset>
```

2. Package `ruleset.xml` into a JAR called `DeploymentRuleSet.jar`.

   ```
   jar -cvf DeploymentRuleSet.jar ruleset.xml
   ```

3. Sign `DeploymentRuleSet.jar` with a valid code signing certificate.

   ```
   jarsigner -keystore jks-keystore -storepass keystore-password -tsa URL for Time Stamping Authority
   DeploymentRuleSet.jar keystore-alias
   ```

4. Copy `DeploymentRuleSet.jar` to *`Windows directory`*`\Sun\Java\Deployment`, for example, `C:\Windows\Sun\Java\Deployment`.

For more information about deployment rule sets, refer to `https://blogs.oracle.com/java-platform-group/entry/introducing_deployment_rule_sets`.

Creating an Exception Site List

If end users are using Java 7 update 51 or above, you can create an exception site list on end user machines to allow users to run Guidewire Document Assistant. Refer to the following URL for instructions:

```
http://java.com/en/download/faq/exception_sitelist.xml
```

Enter the BillingCenter URL for the **Location** field when setting up the exception site list.

Setting Security Levels in the Java Control Panel

You can set the security level in the end user Java Control Panel to Medium to avoid security warnings. Refer to the following URL for instructions:

```
http://www.java.com/en/download/help/jcp_security.xml
```

Guidewire Document Assistant Configuration Parameters

You can set the following configuration parameters related to document management:

Parameter	Description
`AllowDocumentAssistant`	Whether to allow document management controls in the BillingCenter interface. Setting this to `false` removes all controls from the interface, which results in reduced functionality. If `false`, this turns the Guidewire Document Assistant control off entirely and also forces the following parameters to be `false`: • `DisplayDocumentEditUploadButtons` • `UseDocumentAssistantToDisplayDocuments` When Document Assistant is disabled, users can still view documents by downloading the document in the browser. Default: `false`
`DisplayDocumentEditUploadButtons`	Whether the Documents list displays **Edit** and **Upload** buttons. Set this to false if the `IDocumentContentSource` integration mechanism does not support it. Default: `true`
`DocumentAssistantJNLP`	The relative or absolute URL for the Document Assistant JNLP launch file. If specified as a relative URL, the URL is relative to the BillingCenter web server host. Default: `/jnlp/gw/documentassistant/DocumentAssistant.jnlp`
`DocumentContentDispositionMode`	The `Content-Disposition` header setting to use any time that BillingCenter returns document content directly to the browser. The `Content-Disposition` header setting specifies how the browser displays a document. This value can be either `inline` (the default) or `attachment`.

DocumentTemplateDescriptorXSDLocation	Name of the XSD file BillingCenter uses to validate document template descriptor XML files. Specify this location relative to the following directory: modules/configuration/config/resources/doctemplates BillingCenter loads the XSD file from BillingCenter/modules/configuration/config/resources/doctemplates.
MaximumFileUploadSize	Specifies the maximum file size in megabytes that a user can upload to the server. Any attempt to upload a file larger than this fails. Since the server must handle the uploaded document, this parameter protects the server from possible memory consumption problems. Note: This parameter setting affects any imports managed through the BillingCenter Administration tab. This specifically includes the import of administrative data and roles. Default: 20
UseDocumentAssistantToDisplayDocuments	Whether to use the Guidewire Document Assistant control to display document contents. If false, BillingCenter does not use the control and document contents return directly to the browser. Default: true

Document Assistant Supported File Types

The Document Assistant supports auto-launching of the following file types:

• AVI	• GIF	• MDI	• PNG	• RTF	• WAV
• BMP	• HTM	• MOV	• PPS	• RTX	• WMA
• CSV	• HTML	• MPEG	• PPT	• TIF	• XLS
• DOC	• JPG	• MPG	• PPTX	• TIFF	• XLSX
• DOCX	• LOG	• PDF	• PS	• TXT	• XML

For files with extensions not listed, Document Assistant uploads or downloads the file but does not launch the file. Use anti-virus and file system protection software to minimize the risk from downloaded files.

AVI and WMA files can contain security vulnerabilities. Guidewire strongly recommends that you update all client machines with video-playing software that contains the latest security patches. For specific information, see the following Microsoft web pages:

For AVI: http://www.microsoft.com/technet/security/Bulletin/MS09-038.mspx

For WMA: http://www.microsoft.com/technet/security/bulletin/ms09-051.mspx

You can customize the file types that Document Assistant handles by customizing the WhitelistExtensions file within the resources JAR. See “Customizing Document Assistant” on page 151.

Customizing Document Assistant

You can customize Document Assistant by creating a custom version of the resources JAR.

To customize Document Assistant

1. Rename pl-documentassistant-resources__V<*version*>.jar in modules/configuration/deploy/jnlp/gw/documentassistant to some other path under modules/configuration/deploy/jnlp. The new name must be of the form <*base name*>__V<*custom version*>.jar and be within modules/configuration/deploy/jnlp. Place the custom JAR in any subdirectory of modules/configuration/deploy/jnlp,

including gw/documentassistant, so BillingCenter can locate the resource. <*custom version*> must be different from <*version*>. For example, you could rename pl-documentassistant-resources__V8.2.0.jar to pl-documentassistant-customized-resources__V8.2.0-C1.jar.

2. Update DocumentAssistantResources.jnlp for the new custom version.

```
<information>
  <title>Document Assistant Customized Resources <custom version></title>
  <vendor>Guidewire Software, Inc. and <Company Name></vendor>
</information>
...
<resources>
...
  <jar href="pl-documentassistant-customized-resources.jar" version="<custom version>" main="false"/>
...
</resources>
```

3. If the new resource JAR is in a location different from the DocumentAssistantResources.jnlp file, that change must be included as well. For example, if you place the JAR in deploy/jnlp/customized:

```
<resources>
...
  <jar href="/bc/jnlp/customized/pl-documentassistant-customized-resources.jar"
   version="<custom version>" main="false" />
</resources>
```

4. Make changes to the new resources JAR contents, described in "Document Assistant Resource Jar Contents" on page 152. Then re-sign the JAR with a valid code signing certificate specific to your organization using jarsigner or equivalent JDK tools. It is critical that the signature replace any existing signature by Guidewire and be recognized by the end user browsers, or Document Assistant will not load properly.

5. Regenerate the BillingCenter WAR or EAR file to copy your changes to the webapp area. During testing, the Java Control Panel can be used to clear the cache of old copies of changes rather than to constantly iterate on the custom version. Or you can disable caching during testing.

Document Assistant Resource Jar Contents

The documentassistant directory includes the following contents:

File	Description
DocumentAssistant.jnlp	Master JNLP launch file
DocumentAssistantLogo.jpg	Logo image in client Java Control Panel
DocumentAssistantResources.jnlp	Resources Extension JNLP launch file
pl-documentassistant-applet__V<*version*>.jar	JAR containing the Document Assistant applet and internal resources (read-only)
pl-documentassistant-applet__V<*version*>.jar.pack.gz	Pack200 compressed version of Document Assistant applet JAR (read-only)
pl-documentassistant-resources__V<*version*>.jar	JAR containing Document Assistant whitelist extensions and client-side document production scripts. You can customize these resources.

You can create a custom version of pl-documentassistant-resources__V<version>.jar with customized resources. The JAR file includes a documentassistant directory that contains an unknown and a windows directory. The windows directory includes the following resources:

File	Description
ExcelMerge.js	The ExcelMerge.js file contains the JScript that Document Assistant uses when handling an Excel document. You can customize the JScript by modifying this file.
Utility.js	The Utility.js file contains utility scripts used by Document Assistant.

File	Description
`WhitelistExtensions`	A plain-text file containing a dot-delimited list of file extensions that Document Assistant will open. You can add or remove extensions to this file to enable or disable that file type for Document Assistant. Edit the WhitelistExtensions file within the `documentassistant/windows` directory. The single-line compact formatting of the default file is not strictly required. Only the dot delimiter is required. Do not include comments or other extraneous text. Any addition to `WhitelistExtensions` must be added to the MIME configuration on the server. See "Adding a custom MIME type for Document Production" on page 238 in the *Integration Guide*. Removals from WhitelistExtensions do not need to be removed from the server MIME configuration. The server will accept uploads of that MIME type, but the Document Assistant will not automatically open or launch such files.
`WordMerge.js`	The `WordMerge.js` file contains the JScript that Document Assistant uses when handling a Word document. You can customize the JScript by modifying this file.

Disabling Guidewire Document Assistant

BillingCenter disables Guidewire Document Assistant by default. If you enabled Document Assistant, you can later choose to disable it.

The following procedures disable the Guidewire Document Assistant.

To disable the Guidewire Document Assistant

1. Disable Guidewire Document Assistant from the `config.xml` file by setting:
   ```
   <param name="AllowDocumentAssistant" value="false"/>
   ```
2. Save `config.xml`.

chapter 11

Using BillingCenter Server Tools

Guidewire provides **Server Tools** to assist you with certain server and database administration tasks. Typically, users must have the `internaltools` permission to access the **Server Tools** screens. The Superuser role has this permission by default. Alternatively, if the `EnableInternalDebugTools` parameter is set to true in `config.xml`, and the server is running in development mode, all users have access to the **Server Tools** screens. For more information about server modes, see "Server Modes and Run Levels" on page 56.

Press `ALT+SHIFT+T` to display the **Server Tools** tab. At the top of each screen in the title bar is a link that returns you to the main BillingCenter interface. Alternatively, you can **Logout** from this screen to exit BillingCenter.

This topic includes:

Batch Process Info

Use the **Batch Process Info** screen to run and view information about BillingCenter batch processes, including writer threads for work queues. This information includes the **Batch Process** name, **Description**, **Status**, **Last Run** time, **Last Run Status**, **Next Scheduled Run** time, and scheduling information. The **Cron-S M H DOM M DOW** schedule column header stands for seconds, minutes, hours, days of month, month, and day of week.

Note: You can run writers for work queues from the **Work Queue info** screen or the **Batch Process Info** screen.

To run a batch process, click **Run** in the **Action** column for the batch process. The **Run** button is enabled for all batch process types that belong to the `BatchProcessTypeUsage` category `UIRunnable`. To stop a batch process, click **Stop** in the **Action** column for the batch process.

To download a history for a batch process, click **Download History** in the **Action** column for the batch process. You can then specify a range of dates to include in the history information. The historical data includes the date and time that the process started and completed, the number of operations, and the number of failed items along with the failure reason.

Use the drop-down on the **Batch Process Info** screen to filter the batch process list. You can set the filter to show **Any** processes, or only **Schedulable** or **Runnable** processes.

You can use the **Batch Process Info** screen to view the history of a batch process. Select the batch process. The bottom pane provides **Chart** and **History** tabs. The **Chart** tab shows the execution time in seconds and the number of operations performed by the batch process over time. The **History** tab includes a table of records of past runs of the selected batch process. This **History** table includes the following information:

Column	Description
Started	The time that a batch process started.
Completed	The time that a batch process completed.
Ops	For batch processes that are work queue writers, **Ops** is the number of work items processed by a work queue. This number includes work items that failed.
Failed	For batch processes that are work queue writers, **Failed** is a counter that is incremented each time an exception is encountered while processing a work items. The work queue might attempt to process a failed work item multiple times. Therefore, the **Failed** and **Ops** numbers will not necessarily match the total number of work items.
Failure Reason	For batch processes that are work queue writers, **Failure Reason** is the reason that a work item failed processing.

You can configure this screen to restrict the batch processes that can be run from this dialog. To change the configuration, edit the `ServerTools.pcf` screen. Access `ServerTools.pcf` from Guidewire Studio at **configuration** → **Page Configuration** → **pcf** → **tools** → **ServerTools**. The `BatchProcessType` typelist lists the possible batch processes you can display.

Note: You cannot start multiple runs of custom batch processes that are designed to be non-exclusive from the **Batch Process Info** screen. Instead, you must use the maintenance tools command to start multiple runs of non-exclusive custom batch processes.

See also

- “Maintenance Tools Command” on page 188
- “Scheduling Work Queue Writers and Batch Processes” on page 113
- “Exclusive” on page 423 in the *Integration Guide*

Work Queue Info

Use the **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. To work with this dialog, you must first select a **Work Queue**.

From this screen, it is possible to generate and download multiple report types of work queue data:

Report type	Click...	Format	See...
Work queue	**Download**	HTML	Downloading Work Queue Data
Work queue writer runs	**Work Queue Runs**	HTML	Viewing Information on Writer Runs
Work queue instrumentation	**Download Raw Data**	CSV	Downloading Raw Work Queue Data
Work queue history	**Download History**	CSV	Downloading Work Queue History

It is possible to use the statistics from the various reports 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.

Downloading Work Queue Data

Click **Download** to download work queue information. For each report, you can specify the following:

- The maximum number of writers, executors, and batches for each worker
- The number of hours for which to generate item distribution data in the report

To view the report, unzip the download file and double-click `index.html` to open the report.

In general, the report information includes a summary of the work queues and detailed information for specific work queues. The report provides data for each worker by thread and by host, such as:

- How long the worker has been active
- How many items the worker processed
- Throughput (items processed per minute of execution) per thread and per host
- Start and end times for the worker
- Last wake up time
- Items processed per thread (cumulative, average, and maximum)
- Uptime (cumulative, average, and maximum)
- Execution time (cumulative, average, and maximum)

The download file also provides a `_queueruns.html` file. This HTML file provides additional information on the queue runs for the last seven days, for up to 1000 runs. This report also provides links to more detailed information for each work queue.

Viewing Information on Writer Runs

The Work Queue report includes a view called **Work Queue Runs**. This view shows work queue statistics organized by writer run,

including how long it took the writer to produce work items and how long the workers processed those items. The view can be confusing if the writer is run repeatedly before finishing the work items produced by the previous run.

To access this report, open the Work Queue report and click the **Work Queue Runs** link at the bottom of the screen.

Downloading Raw Work Queue Data

To download reports on work queue instrumentation in CSV format, click **Download Raw Report**. The report contains time-sliced raw data from the `ProcessHistory` and `InstrumentedWorkerTask` tables in CSV format for analysis with third-party tools such as Microsoft Excel.

Downloading Work Queue History

To download the history of a particular work queue, first select a work queue. Then, click **Download History** in the **Actions** column of the row for that particular work queue. This action generates a CSV-formatted file that 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 Instrumentation Purge Batch Processing" on page 143

Understanding the Work Queue Table

By default, BillingCenter shows statistics **By Writers** associated with the queue. Use the **By Workers** tab to view the workers associated with the queue.

The top-level columns of the **Work Queue** table have the following meanings:

Column	Description
Work Queue	Name of the work queue.
Available	Number of work items available for processing.
Checked Out	Number of work items checked out by workers.
Failed	Number of work items that failed during processing.
Executors Running	Number of workers processing the work queue.
Writer Status	Status of the writers.
Actions	Actions that you can perform on the work queue. These include: **Run Writer –** Launches the writer to write work items for the work queue. **Notify Executor –** Wake workers by notifying the executor that there are items to process. See "Worker Thread Management" on page 118. **Stop Executor –** Stops the executor currently managing the work queue. **Restart Executor –** Restarts the executor. **Download History –** Downloads the historical instrumentation data for the work queue, in CSV format. You can clear the instrumentation data for all work queues by running the Work Queue Instrumentation Purge process. See "Work Queue Instrumentation Purge Batch Processing" on page 143 for more information.

Understanding Item Statistics

The **Item Statistics** region of the **Work Queue Info** screen provides information on work items. This region has three tabs:

- **By Writers**
- **By Executors**
- **Work Items**

By Writers Tab

The item counts in the **By Writers** columns are the counts generated by the writer. Each row represents one wake period for the writer. These values have the following meanings:

Column	Description
Process ID	The ID for the writer process.
Item Creation Time	The time at which the writer woke and began writing work items. The first item in the table for a queue has a creation time that matches the queue's current **Last Execution Time for the Writer** value.
Scheduled	Whether the writer is scheduled.
Number of Items	The total work items in the queue regardless of status.
Worker End Time	The timestamp when the last worker completed work items.
Execution Time	The number of minutes the process has been executing.
Available	The total number of available items in the queue.
Checked Out	The number of items checked out by workers for processing.
Succeeded	The number of items that completed successfully.
Failed	The number of items that failed.

By Executors Tab

The **By Executors** tab lists active executors. The columns have the following meanings:

Column	Description
Hostname	The server on which the executor is running.
Max. Number of Workers	The maximum number of workers available to the executor.
Processed Items	Number of items processed by the workers.
Exceptions	Number of exceptions encountered during processing.
Failed items	Number of work items that failed processing.
Active	Specifies whether the executor is currently active.
Started	The timestamp when the executor was started.
Up For	The duration that the executor has been running.

Under the **By Executors** tab is a **By tasks** tab. The **By tasks** columns have the following meanings:

Column	Description
ID	The unique identifier of the task.
Writer	The identifier of the writer process.
Success	Whether the processing of work items by the worker was a success.
Checked out items	The number of work items the worker checked out.
Processed items	The number of work items the worker processed.
Exceptions	The number of exceptions, if any, encountered during item processing.

Column	Description
Orphans Reclaimed	The number of orphaned work items the worker has adopted for processing.
Failed items	The number of failed work items.
Skipped items	The number of items that were skipped.
Started	When the task started.
Ended	When the task ended.
Active	Whether the task is still active.
Consecutive Errors	If processing resulted in exceptions, the number of consecutive work items found that resulted in exceptions during processing by the worker.

Work Items Tab

The **Work items** tab lists work items. The columns have the following meanings:

Column	Description
ID	The unique identifier of the work item.
Create time	When the work item was created.
Update time	When the work item was last updated.
Available at	When the work item is available to be processed. This value is null for failed work items.
Server	The server which processed the work item.
Writer	The writer that created the work item.
Attempts	How many attempts a worker has made to process the item.

Set Log Level

Use the **Set Log Level** option to set the logging level for different logging categories. The logging levels you specify persist until you change them or restart the server. To make your settings permanent, edit the `logging.properties` file. Access this file from the **Project** window in Studio by navigating to **configuration** → **config** → **logging**, and the opening `logging.properties`.

Configuration parameters in the `config.xml` file control which logging categories are available on the **Set Log Level** screen. Access this file from the **Project** window in Studio by navigating to **configuration** → **config**, and then opening `config.xml`.

> **Note:** Some log4j loggers do not appear on the **Set Log Level** screen until actually used. This is a standard log4j behavior.

See also

- For information about logging configuration and default logging categories, see “Configuring Logging” on page 21.
- For information on specific parameters, see “Logging Parameters” on page 51 in the *Configuration Guide*.

View Logs

From the **View Logs** screen you can view a log file, filter the log file for specific entries, and set the maximum number of lines to display. By default, BillingCenter writes log files to `tmp/gwlogs/BillingCenter/logs/`. To specify the location where the **View Logs** screen checks for log files, specify the `guidewire.logDirectory` property in the `logging.properties` file.

See also

- "Configuring Logging" on page 21
- "Specifying the Location of Log Files for the View Logs Page" on page 23

Info Pages

The **Info Pages** provide information to help manage a BillingCenter 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. You can access the following **Info Pages**:

- **Configuration**
- **Consistency Checks**
- **Database Table Info**
- **Database Parameters**
- **Database Storage**
- **Data Distribution**
- **Database Statistics**
- **Oracle Statspack**
- **Oracle AWR**
- **Oracle AWR Unused Indexes Information**
- **SQL Server DMV Snapshot**
- **Microsoft JDBC Driver Logging**
- **Load History**
- **Load Integrity Checks**
- **Load Errors**
- **Upgrade Info**
- **Runtime Environment Info**
- **Safe Persisting Order**
- **Loaded Gosu Classes**

Configuration

The **Configuration** screen lists the values of the configuration parameters in your BillingCenter 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`

These files are located in the `config` folder within the downloaded ZIP file. The ZIP file also includes a `current` directory, which includes the in-memory state of `config.xml` and `work-queue.xml` parameters on the server. The in-memory state is made available because certain configuration parameters can be changed using a web service or JMX APIs after server startup.

Consistency Checks

Use the **Consistency Checks** screen to view and run consistency checks on the BillingCenter database. The screen consists of two tabs, which are:

- **Run consistency checks**
- **View consistency checks definitions**

The Run Consistency Checks Tab

Use the **Run consistency checks** tab to submit a batch job to perform database consistency checks. For each consistency check, you can:

- Run the consistency check for all tables, for specific tables, or for a defined table group. To define table groups, see "Defining Table Groups" on page 65 in the *Installation Guide*. You must specify one or more tables.
- Run all default checks or run only specific consistency check types. You must specify one or more consistency check types.
- Optionally specify a **Description**. BillingCenter prepends the description as you view the results to a standard description of the tables and checks.

Running a Consistency Check

To run a database consistency check, select the specific options for the consistency check, then click **Run Consistency Checks**.

After the batch process completes, you can:

- Click the **Download** arrow to download a `ConsistencyCheckRun`*`date`*`.zip` file that contains the set of database reports.
- Click the **View** icon to open a pop-up from which you can view the same reports contained in the `ConsistencyCheckRun`*`date`*`.zip` file, after you supply your user credentials.

After you download the `ConsistencyCheckRun`*`date`*`.zip` file, unzip the file into its own directory. Locate the `index.html` file and double-click it to open it in a browser. You can then use the links on the screen to navigate through the distribution reports.

If any of the consistency check generates a SQL error, BillingCenter adds a **Rerun SQL failures** button next to the consistency check that caused the error. To clear the error, use the report to identify and correct the error. Then, click **Rerun SQL failures** to rerun the consistency check.

Running a Consistency Check Using system_tools

It is also possible to run consistency checks with the `system_tools` command using the `-checkdbconsistency` option. Guidewire provides this option so that you can schedule consistency checks to run asynchronously during the times that the database is handling fewer requests. See "Running Consistency Checks with System Tools" on page 37 and "System Tools Command" on page 191 for more information.

The View Consistency Checks Definitions Tab

The **View consistency checks definitions** tab lists the consistency checks available for each database table and provides a description of each check. In this tab, you can:

- Search for consistency check types to see a list of all tables for which that consistency check is available.
- Search by table name to find the consistency checks related to that table. Most consistency checks operate on the specified table, but some checks, such as typelist table checks, operate on other tables as well.
- Select a table name from the list to view a read-only version of the SQL query that generates the consistency check.
- Click **Download** to download a ZIP file that contains HTML files describing all of the consistency checks provided in the BillingCenter base configuration.

Viewing Downloaded Consistency Check Information

To view the downloaded information, extract the ZIP file and open the `index.html` file. From the `index.html` file or the **View consistency checks definitions** tab you can do the following:

- Click **Table Name** to sort consistency checks by table name.

- Click **Check Name** to sort consistency checks by check name.
- Select the **Command** tab to view the SQL command of the consistency check. The SQL command retrieves a count of rows that violate the consistency check.
- Select the **Query to identify rows** tab to view the SQL query used to identify rows that violate the consistency check. SQL queries to identify rows that violate consistency checks are not available for all check types.

From the `index.html` file only, you can do the following:

- Click a table name to view all consistency checks related to that table.
- Click a check name to view all tables that the consistency check runs against.

Database Table Info

Use the **Database Table Info** screen to download index and key information for each table and to verify the database schema against the data model. These screens document the names of the BillingCenter generated indexes and constraints. The following screens are provided in the download:

Screen	Description
All Tables	Provides information about all BillingCenter tables.
Guidewire Version	Lists schema version and build information for BillingCenter.
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.
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 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.
XML Configuration Files	Click **config_files: Directory with config files** to access a listing of the XML database configuration files.

If your database is receiving integrity check errors or referential integrity problems, Guidewire Support might ask you to download the information from this screen and provide it to them.

You can click **Verify** to have BillingCenter compare the database schema with the schema defined in the data model files. You can download a report of schema verification errors by clicking

Database Parameters

The **Database Parameters** screen displays information about the database configuration. A drop-down menu provides a list of database parameter types that you can view on this screen. The following selections are available:

Tab	Description
Database and Driver	The versions of the database and its associated driver.
Database Connection Pool Settings	Connection pool settings as configured in `config.xml` if using BillingCenter to manage the connection pool. See "Configuring Connection Pool Parameters" on page 33 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.
Guidewire Database Config	Guidewire-specific database configuration parameters. BillingCenter reads these parameters from the `database` block in `config.xml` or uses a default value if `config.xml` does not specify database parameters.
Guidewire Database Config Statistics Settings	Guidewire-specific database configuration parameters related to statistics gathering.
Guidewire Database Upgrade Configuration	Guidewire-specific database configuration parameters related to upgrade.
Database Connection Properties	Properties related to the database connection, including whether **Autocommit** is on, the **Transaction isolation level** and whether the database connection is **Read Only**. This does not include the JDBC URL or credentials information. BillingCenter shows the JDBC URL under **Database Connection Pool Settings**.
SQL Server Server Global Server Settings	SQL Server only. This view describes global server settings for the SQL Server instance.
SQL Server Database Options	SQL Server only. This view shows options set on the SQL Server database, such as `auto create statistics` and `auto update statistics`, the recovery model, collation, and so forth.
SQL Server Server Instance Attributes and Values	SQL Server only. This view shows attributes and values for the SQL Server instance to which BillingCenter is connected.
SQL Server Session Properties	SQL Server only. This view shows properties of the SQL Server session for the current connection with BillingCenter.

You can click **Download Database Parameters Info** to download an HTML file containing all database parameters.

If you troubleshoot a database performance issue with Guidewire Support, Guidewire Support might ask you to send this parameter information as a reference. Integration consultants also use this data while tuning database performance.

Database Storage

The **Database Storage** screen provides information about the space and memory taken up by the database on the BillingCenter server. You can view and download database storage information.

The following tables lists the filtering options for the database storage information:

Option	Description	Database type
Include Estimation of Compression Savings for Tables and Indexes	If checked, you need to also select the compression level for estimating savings.	Oracle SQL Server
Select Compression Level for Estimating Savings	Only available if you chose to include estimation of compression savings in the database storage information. Options include: • Oracle – Advanced • SQL Server – Screen, Row	Oracle SQL Server
Collect Index Physical Stats for all tables	If you select **Yes**, then BillingCenter includes statistics on all tables in the database. If you select **Specify tables**, BillingCenter includes statistics only on the database tables that you select.	SQL Server
Select Mode for Collecting Index Physical Stats	Options include: • none • Detailed • Limited • Sampled	SQL Server

After setting the desired filtering options, chose one of the following:

- To see the database storage information on the current BillingCenter 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.

IMPORTANT Guidewire recommends that you download and save the database storage information right before and immediately after an upgrade or other significant database change. This provides you with a point of reference that you can provide to Guidewire Support if requested.

Oracle Database Options

The **Storage Set to Display** drop-down filter has the following options for Oracle databases:

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.

Storage Set to Display	Description
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, BillingCenter 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. BillingCenter 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 used or 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

The **Storage Set to Display** drop-down filter has the following options for SQL Server databases:

Storage Set to Display	Description
Data Spaces	Lists the filegroups taken up by the data and the amount of space taken and allocated by BillingCenter.
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.
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. 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, BillingCenter 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. BillingCenter 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.

Data Distribution

Use the **Data Distribution** screen to a run batch processing job that generates data on the distribution of various items in the database. To generate the data report:

1. Select from the available options listed under **Data Distribution Batch Job Parameters**.

2. (Optional) Select the **Specify tables** option if you want to limit or filter the number of database tables in the report.
3. Click **Submit Data Distribution Batch Job**.

After the batch job completes, you can:

- **Download the report** – Click the **Download** arrow to download a `DataDistribution.zip` file that contains the set of database reports.
- **View the report** – Click the **View** icon to open a pop-up from which you can view the same reports contained in the `DataDistribution.zip` file, after you supply your user credentials.

Viewing the Downloaded Data Distribution Report

After you download the `DataDistribution.zip` file, unzip the file into its own directory. Locate the `index.html` file and double-click it to open it in a browser. You can then use the links on the screen to navigate through the distribution reports.

Downloading Comparison and Combined Data Distribution Reports

To create either a comparison or a combined report, two or more data distribution reports and their output must exist in the database.

To download a ZIP file of a comparison or combined report:

1. Select (check) at least two of the generated reports.

   BillingCenter enables the following download buttons:

   - **Download Comparison Zip File**
   - **Download Combined Zip File**
2. Click the appropriate button.

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.

Running Data Distribution Batch Processing from the Command Prompt

It is also possible to start the data distribution batch process directly from the command prompt by using the `maintenance_tools` command. To do so, enter the following at the command prompt:

```
maintenance_tools -password password -startprocess datadistribution
```

See “Maintenance Tools Command” on page 188 for details.

Database Statistics

The **Database Statistics** screen provides reports about out-of-date statistics in the database indexes, histograms, staging tables, and BillingCenter tables. The **Database Statistics** screen contains two tabs:

- **DatabaseStatisticsInfo**
- **Execution History**

For information about generating the database statistics, see “Configuring Database Statistics” on page 39.

The DatabaseStatisticsInfo Tab

Use the **DatabaseStatisticsInfo** tab of the **Database Statistics** screen to view database statistics reports for the entire database or for specific tables. This tab presents you with the following options:

- **View database catalogs statistics on all tables** - To specify tables on which to gather statistics, first select **No**. Then select the checkbox next to each table for which you want statistics. If you do not select tables from the list, BillingCenter reports statistics information from the database metadata only.
- **Show Previous Statistics** - To view database statistics from previous points in time, select **Yes.**

Click **Download** to download the statistics report.

BillingCenter database statistics reports include the following information.

Tab	Description
Unanalyzed Indexes	Lists the indexes that were not broken out for statistical purposes.
Unanalyzed Histograms	Lists the histograms that were not broken out for statistical purposes.
Stale Index Stats	Details potentially out-of-date indexes on BillingCenter tables. BillingCenter considers the statistics of an index as potentially out-of-date if the estimated row count does not match the actual row count. However, it is not uncommon for the estimates to differ from the actual numbers without there being a problem. To confirm if the statistics are out-of-date, you or your company's DBA must perform an analysis.
Stale Histogram Stats	Details potentially out-of-date histograms on BillingCenter tables. The report considers histogram statistics as potentially out-of-date if the estimated row count does not match the actual row count. However, it is not uncommon for the estimates to differ from the actual numbers without there being a problem. To confirm if the statistics are out-of-date, you or your company's DBA must perform an analysis.
Application Tables	Displays statistics for individual BillingCenter tables. Use the **Pick table to display stats** drop-down to select a table. The drop-down lists the current row count for each table. After you select the table, you can view the data associated with indexes and histograms (if defined) on the table.
Staging Tables	Displays statistics for individual staging tables. Use the **Pick table to display stats** drop-down to select a table. The drop-down lists the current row count for each table. After you select the table, you can view the data associated with indexes and histograms (if defined) on the table.
Typelist Tables	Displays statistics for individual typelist tables. Use the **Pick table to display stats** drop-down to select a table. The drop-down lists the current row count for each table. After you select the table, you can view the data associated with indexes and histograms (if defined) on the table.

BillingCenter provides database statistics generation designed specifically for how the BillingCenter application and data model interact with the physical database. Generating database statistics from the database management system can potentially create statistics that cause BillingCenter to select a bad plan for execution of SQL queries against the database. Therefore, always use BillingCenter to generate database statistics, rather than by using the statistics generation provided with the database management system.

The Execution History Tab

The **Execution History** tab of the **Database Statistics** screen lists the time at which the command to update database statics was run. In development mode, you can also generate full or incremental database statistics directly from the **Execution History** tab of the **Database Statistics** screen.

Oracle Statspack

This screen is available only if the database server is Oracle. To display statspack information, you must have installed the statspack option in your Oracle database. Refer to Oracle documentation for instructions. You must also create statspack snapshots by using a tool such as SQL*Plus or SQL Developer. The **Oracle Statspack** screen displays statspack snapshots you have created. A snapshot gives you database configuration and performance statistics for the duration you defined while creating the snapshot.

Oracle AWR

The Oracle AWR information screen is available only if the database server is Oracle. The Oracle Automatic Workload Repository (AWR) is an upgrade of the information available with the Oracle statspack. Refer to Oracle documentation for details.

Use the **Oracle AWR** screen to generate a set of performance reports using AWR snapshots that you define in the database. You must select two snapshots that share the same Oracle instance startup time. The report contains information on the Oracle database tables, database parameters, and database table statistics.

After BillingCenter completes generating the report, you can:

- Click the **Download** arrow to download an `AWRReport.zip` file that contains the set of database reports.
- Click the **View** icon to open a pop-up from which you can view the same reports contained in the `AWRReport.zip` file, after you supply your user credentials.

You can also retrieve performance reports based on Oracle AWR snapshots from the command prompt using the `system_tools -oraPerfReport` option.

The `system_tools -oraPerfReport` 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 191
- “Maintenance Tools Command” on page 188

Oracle AWR Unused Indexes Information

This screen is available only if the database server is Oracle. The **Oracle AWR Unused Indexes Information** screen enables you to select two snapshots from the same instance startup time. Select snapshots that have a wide range. The downloadable report provides information about indexes that have no logical or physical reads or are not found in query plans.

SQL Server DMV Snapshot

The **SQL Server DMV Snapshot** screen is available only if the database server is SQL Server. Use this screen to generate and download performance reports using SQL Server Dynamic Management Views. You can optionally specify whether to **Include Database Statistics** in the report. Click **Generate Perf Report** to launch an internal batch process that gathers performance data and creates the report.

The **SQL Server DMV Snapshot** screen lists the generated reports in a table with **Download and View** for each report:

- Click the **Download** arrow to download an `DMVReport.zip` file that contains the set of database reports.
- Click the **View** icon to open a pop-up from which you can view the same reports contained in the `DMVReport.zip` file, after you supply your user credentials.

After you download the `DMVReport.zip` file, unzip the file into its own directory. Locate the `index.html` file and double-click it to open it in a browser. You can then use the links on the screen to navigate through the distribution reports.

It is also possible to generate a SQL Server DMV report using the `-mssqlPerRpt` option of the `system_tools` command.

See also

- “System Tools Command” on page 191

Microsoft JDBC Driver Logging

The **Microsoft JDBC Driver Logging** screen is available only if the database server is SQL Server. Use this screen to set the **Logging Level** for the Microsoft JDBC driver. Be cautious setting the logging level, as detailed logging can slow the system significantly. You can specify the **Logging Format** as **Simple**, for a more readable format, or **XML** for XML output, usually parsed by another system. You can also specify the **Log File Location**, including special components that are replaced at runtime, such as `%u` to append a unique number to each log to avoid conflicts.

Load History

The **Load History** screen displays information about database load operations that completed in BillingCenter. These load operations execute as you import data into the database. For example, loading data into the staging tables impacts the loader history information.

This screen contains a summary view and a detail view. The summary view lists information about each specific load operation such as who called it, when, how long the operation took, any errors generated, and so forth. You can drill down into the details of an operation by clicking **View.**

The detail view shows on two tabs the **Steps** in the operation and the impact of the operation on **Row Counts.** You can drill down into the individual **Steps** by clicking on the step. Use the **Row Counts** screen to quickly assess whether the amount of data that the operation loaded was the amount that you expected the operation to load.

See also

- "Importing from Database Staging Tables" on page 367 in the *Integration Guide*

Load Integrity Checks

The **Load Integrity Checks** screen reports on the SQL integrity checks that run as a database load operation executes. This screen has two view:

- **View by Staging Table**
- **View by Load Error Type**

For each integrity check, the **Load Integrity Checks** screen lists the SQL query that the check performs. The **Load Integrity Checks** screen also lists a description of the check and the associated load error type or staging table.

In both views, you can enable **Allow Non Admin References**. If you enable them, BillingCenter checks foreign key references to administrative tables on load, such as users and groups. These references are disabled in the base configuration of BillingCenter.

See also

- "Data Integrity Checks" on page 389 in the *Integration Guide*

Load Errors

The **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, BillingCenter cannot identify or store a single LUWID for the error.

See also

- "Data Integrity Checks" on page 389 in the *Integration Guide*

Upgrade Info

The **Upgrade Info** screen displays information about the automatic upgrade that runs upon server startup. You can use this screen to see what steps the upgrader ran and the impacts to row counts and storage information.

In the **Upgrade Info** table:

- Click the **Download** arrow to download a `UpgradeInfo.zip` file that contains a set of HTML reports describing various aspects of the upgrade process.
- Click the **View** icon to open a pop-up from which you can view the same reports contained in the `UpgradeInfo.zip` file, after you supply your user credentials.

After you download the `UpgradeInfo.zip` file, unzip the file into its own directory. Locate the `index.html` file and double-click it to open it in a browser. You can then use the links on the screen to navigate through the distribution reports.

File `UpgradeInfo.zip` contains several different types of 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.
- **Database Parameters** – Lists information on various database parameters, including the database connection pool settings, the database configuration settings, and similar information.

See also

- "Understanding and Authorizing Data Model Updates" on page 35
- "Viewing Detailed Database Upgrade Information" on page 182 in the *Upgrade Guide*

Runtime Environment Info

The **Runtime Environment Info** screen lists information about the runtime environment for BillingCenter. This information includes Guidewire platform and BillingCenter build information, system properties, and environment variables.

Safe Persisting Order

The **Safe Persisting Order** screen lists the order in which BillingCenter runs the Preupdate rules for their root entities.

Loaded Gosu Classes

The **Loaded Gosu Classes** screen provides a list of all the Gosu classes that have been loaded by BillingCenter.

Management Beans

BillingCenter includes management beans that represent different resources. You can use this dialog to view all and edit some of the attributes associated with these resources. The resources available from the dialog include:

Resource	Description
`com.guidewire.pl.system.monitor`	View session and connections information associated with the BillingCenter application server. If a user is logged in more than once, the user name has the number of sessions appended to it in parentheses.

com.guidewire.pl.system.configuration	View all and edit some of the configuration values in the system. You must have the soapadmin permission to change values associated with management beans. Configuration parameters that you can edit from this screen have editable **Value** fields. After you edit a value, BillingCenter enables a **Save** and **Cancel** button. You must save your edit before it becomes active.
com.guidewire.pl.system.cluster	View information about the servers in a cluster. This bean is only available if you run the server in a clustered environment.
com.guidewire.pl.system.cache	View all and edit some cache attributes. You must have the soapadmin permission to change values associated with management beans. Cache parameters that you can edit from this screen have editable **Value** fields. After you edit a value, BillingCenter enables a **Save** and **Cancel** button. You must save your edit before it becomes active.

See also

- "Application Configuration Parameters" on page 27 in the *Configuration Guide*
- "Clustering Application Servers" on page 73
- "Cluster Info" on page 173
- "Cache Info" on page 173
- "Application Server Caching" on page 63

Viewing and Changing Caching Configuration Values

For com.guidewire.pl.system.cache, you can set the caching configuration values:

- MaxCacheSpace – Maximum amount of memory to use for the cache
- StaleTimeMinutes – Number of minutes an object can exist in the cache before being refreshed.

If an object remains in the cache longer than StaleTimeMinutes, BillingCenter refreshes the object from the database. Monitor cache performance from the **Cache Info** screen. Then, adjust cache values based on your analysis of that information.

See also

- "Cache Info" on page 173
- "Application Server Caching" on page 63

Startable Plugin

The **Startable Plugin** screen lists the name and status of each startable plugins registered for this Guidewire installation. From this screen, you can start or stop each listed plugin.

See also

- "Startable Plugins" on page 273 in the *Integration Guide*

Cluster Info

The **Cluster Info** screen provides information on the clustered environment, if clustering is enabled. To view this screen, configuration parameter `ClusteringEnabled` must be true. It is possible to access this screen from any application server node in the cluster.

The **Cluster Info** screen provides information on the following:

Area	Lists	Provides information on...
This Application Server Instance	• Host name • Server ID • Logical name	The server node on which you view the **Cluster Info** screen. • *Host name* – Machine name for this server node. • *Server ID* – Guidewire designated name. Specify either through the cluster `<registry>` element in `config.xml`, or, through a JVM option at server start up: `-Dgw.bc.serverid=`*`serverNode`*`.` If there is no specified server ID, then BillingCenter uses the host (machine) name as the server ID. • *Logical name* – JGroups name for this server node.
Batch Server	• Server ID • Logical name	The current batch server, if one exists. To make the current server node the batch server node, click **Promote to Batch Server**.
Cluster Members	• Server ID • In cluster now • Logical name • Run level • Server started • Connection started • Last update	The individual member nodes currently recognized by the cluster. To update this information, click **Refresh Cluster Info**. A review of this information is one way to determine if a cluster node has become unreachable, and, therefore, missing from the list.
History	• Logical name • Last run level • Server started • Server stopped	The history of all member nodes in the cluster.

Downloading a Cluster Server Report

Clicking **Download** opens a **Configure Cluster Info** report tab. To generate the report:

1. Select the **Include server history in the report** option if you want to include this information in the generated report.
2. Set a value for **Maximum number of history records (for each server)** option. As indicated, the value that you set applies to the maximum number of records to report for each server node in the cluster.
3. Click **Complete Download**.

The report generator creates a ZIP file that contains the actual report. To view the report, open `index.html`. If the report includes server history, click the **Server ID** link for each cluster member to open its history report.

Cache Info

The **Cache Info** screen provides graphical representations of BillingCenter application server cache information to help you monitor how well the cache is performing.

This topic includes:

The Cache Summary View

The **Cache Summary** view on the **Cache Info** screen includes graphs of the following:

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 BillingCenter evicting an object because the cache was full and the number of missed caused by BillingCenter evicting an object due to reaping. 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 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.

Click **Edit** to modify the maximum cache space and stale time parameters from the **Cache Summary** view. If you change these parameters from the **Cache Summary** view, the values you specify apply only to the application server node to which you are connecting. If you restart the server, your changes are lost. For your changes to persist, edit the their values `config.xml` file.

Click **Download** to download a CSV file containing detailed cache information.

Click **Clear Global Cache** to clear the cache entirely of all entities. The cache always contains some objects to support an active application server.

Click **Refresh** to display the most current information.

See also

- “Application Server Caching” on page 63

The Historical Performance View

The **Historical Performance** view on the **Cache Info** screen includes graphs of the following:

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.

Graph	Description
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 BillingCenter having evicted an object from the cache because the cache was full.

Click **Refresh** to display the most current information.

The Cache Details View

The **Cache Details** view on the **Cache Info** 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.

Click **Refresh** to display the most current information.

Guidewire Profiler

Guidewire Profiler is a set of tools using a common interface. The Profiler can gather and help analyze information on the runtime behavior and performance of Guidewire BillingCenter. 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.

There are several concepts that are important to an understanding of how to configure application profiling. They are:

- **Profiler stack** – A profiler stack stores profiling information for a specific thread. See “Profiler Stacks” on page 180 for more information.
- **Profiler frame** – A profiler frame contains information corresponding to a specific invocation of profiled code, such as its start and finish times. See “Profiler Frames” on page 180 for more information.
- **Profiler entry point** – A profiler entry point refers to the type of interaction that you select to profile. You enable and disable the different types of profiling in the **Configuration** screen of Guidewire Profiler. See “Profiler Entry Points” on page 176 for more information.

  Note: Guidewire Profiler does not collect memory usage statistics. You can use a third-party tool to gather memory usage and garbage collection information.

This topic includes:

See also

- “Analyzing Server Memory Management” on page 68

Guidewire Profiler: Configuration

You configure application profiling on the **Configuration** screen of Guidewire Profiler.

Profiler Entry Points

The 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.

Entry point	Description
Web	Application user interface in which you can perform tasks and actions. Clicking around in the Guidewire application in a browser potentially causes rules, web services, messages, and similar items to trigger. See "Web Session Profiling" on page 178 for more information.
Batch process	Batch processes wake up at regular intervals, and can execute large queries. See "Batch Processing" on page 107.
Work queue	Long running processes that pick up work to be done from a queue. These processes are typically distributed across several servers. See "Batch Processing" on page 107.
Message destination	Process that delivers messages to one or more destinations. See "Messaging and Events" on page 303 in the *Integration Guide*.
Web service	SOAP requests received by BillingCenter. See "Web Services" on page 25 in the *Integration Guide*.
Startable plugin	Plugin implementations that BillingCenter initializes at server startup and de-initializes at server shutdown. For example, it is possible to register a plugin implementation as a listener on a JMS queue. See "Startable Plugins Overview" on page 273 in the *Integration Guide*.

Except for Web entry points, Guidewire Profiler stores configuration information in the database. This information is visible to all application servers in the cluster. Note that any changes to a configuration will take some time to propagate through the cluster. Also, it may take up to the cache stale time for a 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, BillingCenter persists this data database, making it possible to retrieve the data later.

Enabling/Disabling Entry Point Profilers

To initiate application profiling, you enable the specific types of entry points that you want to profile.

To see the results of a profiling session, navigate to the specific entry point link in the left-hand navigation pane of **Guidewire Profiler** → **Profiler Analysis**. See "Guidewire Profiler: Profiler Analysis" on page 178.

Web Profiling

The **Configuration** screen of Guidewire Profiler contains an area at the top of the screen labeled **Web Profiler.** You use this area to set the tracing options that want for this Web profiling session. Click **Enable Web Profiling for this Session** to start application Web profiling. For more information about Web profiling, see "Web Session Profiling" on page 178.

Other Entry Point Profiling

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. 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. Selecting a destination and clicking **OK** returns you to the **Configuration** screen. BillingCenter then adds a message destination row to the **Entry Point Configuration** table, with a different column for each profiling option.

If a profiling option is available for the chosen entry point, BillingCenter places a clickable icon in the table cell for that option. Initially, BillingCenter disables all of the available options (the table cell shows a red X). To enable the option, click the icon, which then changes to a green check mark.

For each entry point row, BillingCenter also adds an **Edit Configuration** button at the right-hand end of the row. Clicking this button provides an alternative way to enable tracing options, as well as a way to disable tracing for this entry point.

See also

- "Profile Tracing Options" on page 177
- "Guidewire Profiler: Profiler Analysis" on page 178

Profile Tracing Options

It is possible to enable various different types of tracing options. 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

To enable a specific profiling option, select or check the box next it.

Tracing option	Description
Stack Trace Tracking	Captures the Java stack trace and the PCF trace at the point at which a query executes. Use caution with **Stack Trace Tracking** as this tracing option 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 only. This trace indicates how the database arrived at a specific execution path. This creates a trace file on the database server.
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.
High Resolution Clock	Microsoft Windows operating system only. 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.
Diff DBMS Instrumentation Counters	Oracle only. Enable this option to capture the DBMS counters at the beginning of the profiling session and to include analysis of the differences in the DBMS-specific report.
DBMS Instrumentation Capture Threshold for each Action (millis)	Oracle only. The Profiler generates a DBMS report if an action exceeds the threshold value set by this option. 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.

Web Session Profiling

To enable Web profiling, click **Enable Web Profiling for this Session** on the **Configuration** tab. It is only possible to enable profiling for Web entry points for the current session.

After enabling Web profiling, BillingCenter records all subsequent round-trips to the server as a separate profiler stack. Unlike other types of entry points, BillingCenter does not persist the stacks from Web requests to the database. Instead, BillingCenter 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 BillingCenter 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.

To use Web profiling

1. Navigate to the application screen that you want to profile.
2. Press ALT+SHIFT+P to open Guidewire Profiler.
3. Select the profiling options that you want to use for this profiling session.
4. Restart the associated batch process, work queue, or message destination, if appropriate.
5. Click **Enable Web Profiling for this Session**. This action returns you to the application screen from which you started.
6. Exercise the application screens that you want to profile.
7. Press ALT+SHIFT+P to return to Guidewire Profiler.
8. Click **Disable Web Profiling** to end the current Web profiling session and generate the **Web Profiler** screen.

To understand the **Web Profiler** screen, see “Guidewire Profiler: Profiler Analysis” on page 178.

You can also download and save this snapshot of application profiling data. See “Downloading and Viewing Profiler Data” on page 179 for details.

Guidewire Profiler: Profiler Analysis

After you disable the current application profiling session, BillingCenter generates a **Profiler Analysis** screen, or, in the case of Web profiling, a **Web Profiler** screen. To see the results of a profiling session, navigate to the specific entry point link in the left-hand navigation pane of **Guidewire Profiler** → **Profiler Analysis**.

The **Profiler Analysis** screen contains multiple 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**.

View Type	Result
Elapsed	Lists each frame in chronological order within its stack along with the time in seconds between when BillingCenter pushed and then popped the frame.
Chrono	Lists each frame in chronological order within its stack along with the time in seconds between BillingCenter creating the stack and pushing the frame. See "Viewing Rule Information in the Profiler Chrono Report" on page 43 in 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** pane contains the message "No profiler stacks found". See "Generating a Profiler Rule Execution Report" on page 42 in the *Rules Guide*.

It is also possible to download a snapshot of the profiling data by clicking the **Download** button. See "Downloading and Viewing Profiler Data" on page 179 for more information.

Downloading and Viewing Profiler Data

To save the current session of Profiler data, click **Download** in the **Profiler Result** region of the **Profiler Analysis** screen. This action 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.

1. Open Guidewire Profiler and navigate to **Profiler Analysis** → **Saved File**.
2. In the **Restore Snapshot** field, browse to find the downloaded file.
3. Click **OK**.

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.

Guidewire Profiler: Tags, Frames, and Stacks

The following sample code introduces key concepts for understanding Guidewire Profiler:

In `ProfilerTag.java`:

```
ProfilerTag MYTAG = new ProfilerTag("MyTag");
```

In `MyCode.java`:

```
import gw.api.profiler.Profiler;
ProfilerFrame frame = Profiler.push(ProfilerTag.MYTAG);
try {
  myMethod(str);
} finally {
    Profiler.pop(frame);
}
```

Profiler 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. In this example, the profiler tag has the name `MyTag` and corresponds to the code invoked by the method `myMethod`.

It is better to create a static final `ProfilerTag` and preserve it, rather than create one each time you need it. Creating the tag incurs some slight performance cost.

Profiler 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`.

Properties and Counters on a Frame

Profiler frames can hold user-defined properties and counters that provide more information about system events. Consider this example:

```
frame.setPropertyValue("PARAMETER", str);
int ret = myMethod(str);
frame.setCounterValue("RETURN", ret);
```

After the coder pops a profiler frame off the stack, the frame contains information about which parameter was passed to `myMethod()` and the return value. Currently, properties and counters are used, among other things, to record SQL being executed, parameters to that SQL, row count returned, which rule is being executed, and so forth.

Note: Exercise care when using this feature. Storing too much information will cause the display to become too cluttered, require more space for storage and, for long-running processes, hold on to too much memory at runtime.

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 is currently being executed. 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 Guidewire Profiler with Custom Code

You can profile your custom code by using Guidewire Profiler.

Creating New Profiler Tags

Define a new profiler tag to associate with the code that you want to profile. To do this, create a globally-accessible Gosu class that extends `gw.api.profiler.BCProfilerTag`. Within this class, define all of your custom profiler tags. Define these tags as constants. For example:

```
static final ProfilerTag MYTAG = new ProfilerTag("MyTag");
```

You use this class to describe the piece of code that you want to profile. put all their profiler tags as constants (static final) in one globally-accessible class.

To profile a block of your custom code, use the following pattern to push and pop profiling information onto the profiler stack. This code works for both Gosu and Java.

```
ProfilerFrame frame = Profiler.push(ProfilerTag.MYTAG);
try {
  /// CODE YOU WANT TO PROFILE
} finally {
Profiler.pop(frame);
}
```

Profiling spawned threads

Some processes spread their workload across multiple threads. If you want to profile those threads, use the following pattern:

```
gw.api.profiler.Profiler.createPotentiallyProfiledRunnable(ProfilerTag entryPointTag,
    String entryPointDetail, GWRunnable block)
```

This generates a new `Runnable` 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.

Using 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

- “Profiling Web Services” on page 131 in the *Integration Guide*.

Funds Tracking

The **Funds Tracking** page shows the current status of Funds Tracking. The page also provides a means to enable or disable this functionality.

chapter 12

Using BillingCenter Internal Tools

WARNING Guidewire does not support the **Internal Tools**. Guidewire provides these tools for use during development only. Guidewire does not support the **Internal Tools** for production environments. Use these tools at your own risk.

The **Internal Tools** page is only available if the server is in development mode. You can put the server in development mode by setting the JVM parameter `-Dgw.server.mode=dev`. Users with the `internaltools` permission can then access the **Internal Tools** pages by pressing `ALT+SHIFT+T` and selecting the **Internal Tools** tab. The **Superuser** role has the `internaltools` permission by default. For more information about server modes, see "Server Modes and Run Levels" on page 56.

This topic includes:

Reload

The **Reload** page is useful while you develop a configuration. From this page you can reload key configuration files into a running BillingCenter 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, BillingCenter writes the errors to the log.
Verify All PCF Files	Verifies the PCF files without reloading them.
Reload Web Templates	Reloads the entire BillingCenter user interface including the `config/web/templates` directory.

Reload Workflow Engine	Reloads the Workflow engine.
Reload Display Names	Reloads label definitions only from the `display.properties` for the locale.

System Clock

The system clock plugin, `TestingClock`, enables you to get and set the current system time in BillingCenter.

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 can not 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 BillingCenter server is in development or test mode.

You must implement and configure the `TestingClock` plugin to use this tool.

See also

- See "Testing Clock Plugin (Only For Non-Production Servers)" on page 266 in the *Integration Guide* for implementation details.

BC Sample Data

The **Sample Data** page is for loading sample data into BillingCenter for development purposes only. Guidewire does not support this tool for a production environment. See "Installing Sample Data" on page 52 in the *Installation Guide* for instructions.

Accounting Config

The **Accounting Config** page lists the **T-Accounts**, **Transactions**, and **Charges** configured in BillingCenter.

chapter 13

Using BillingCenter Command Prompt Tools

BillingCenter includes a number of administrative tools as command prompt tools that you can use for help with administrative tasks on your BillingCenter server.

This topic includes:

See also

- For tools that build BillingCenter, see "Build Tools" on page 99 in the *Installation Guide*.

Administration Tools Overview

BillingCenter provides a set of tools that you can use to perform administrative tasks directly from the command prompt. Typically, these commands are meant to run on an administrator's workstation. The tools are all found in the `BillingCenter/admin/bin` directory, unless otherwise noted. These tools all execute against a running BillingCenter 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 BillingCenter server is actively running.

The following table provides a summary of what each tool does.

Command name	Description
`data_change`	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 Customer Support. Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting the data change on a production server.
`import_tools`	Set of utilities for loading XML-formatted data into BillingCenter.
`maintenance_tools`	Set of utilities for performing maintenance operations on the server (for example, running escalation/exception rules, calculating statistics, and more.)
`messaging_tools`	Provides a set of utilities for managing integration event messages (for example, retrying a message, skipping a message, purging the message table, and more).
`system_tools`	Provides a set of utilities for controlling the server (for example, pinging the server, bringing the server in and out of maintenance mode, updating database statistics, and more.)
`table_import`	Used for importing tables into the database.
`workflow_tools`	Allows you to manage user workflows in the system.
`zone_import`	Loads zone data from a file to a staging table.

Accessing Tool Help

To access help for any tool, enter `-help` after the tool name. For example, enter the following at the command prompt to generate a list of tool options with a description of each option for the `import_tools` administrative tool:

```
C:\guidewire\bc804\admin\bin>import_tools -help
```

Administrative Tool Command Syntax

The administrative tools command descriptions use the following command syntax.

tool_name	Bold font indicates that this is the actual command name, for example, `import_tools`.
`-option`	All tool options start with a minus sign (-). Tool 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.

Data Change Command

```
data_change -help
data_change -password password [-server url] [-user user] {
            -edit refid -gosu filepath [-description description] |
            -discard refid |
            -status refid |
            -result refid }
```

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

The user who runs this command must have permission `wsdatachangeedit`.

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

See also

- For a description of how and when to use the `data_change` command to change data on a running production server, see "Data Change API" on page 47.
- For a description of the `data_change` command options, see "Data Change Command Prompt Reference(data_change.bat)" on page 51.
- For a description of how to use the `DataChangeAPI` web service, see "Data Change Web Service Reference (DataChangeAPI)" on page 52.

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 BillingCenter database. You can only import data for valid entities or their subtypes. BillingCenter supports this command for importing administrative data but not for importing other data into BillingCenter. Instead, use staging tables or APIs to import other types of data into BillingCenter.

Note: BillingCenter does not fire any events related to the data you add or modify through this command.

Data that you import into BillingCenter through the use of `import_tools` is immediately available. There is no need to restart the application 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 92
- "Understanding the import Directory" on page 92
- "Importing Administrative Data Using the import_tools Command" on page 100
- "Importing Administrative Data" on page 127 in the *Integration Guide*

Import Tools Options

You can use any of the following options with the `import_tools` command. You must always supply the `-password` option.

Option	Description
`-charset` *charset*	Character set (*charset*) in which the files for import are encoded. If this option is null, the default character set encoding is UTF-8.
`-dataset` *integer*	Integer value (*integer*) representing the dataset to import from a CSV-formatted file, for example: `RolePrivilege,0,`**default_data:1**`,acctchargesview,billing_clerical` Datasets are ordered by inclusion. The smallest dataset is always numbered 0. Thus, dataset 0 is a subset of dataset 1, and dataset 1 is a subset of dataset 2, and so forth. To import all data, set this value to -1.
`-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 either a CSV file (a comma-separated list of data) or an XML files. It is possible to provide the list of file names in a separate file. To do so, create a file with a comma-separated list of files names. Pre-pend @ to the name of the list file that you create, for example: `-import @files.lst`
`-output_csv` *filename*	If used with the `-import` option, outputs comma-separated values to the specified file and then stops processing. BillingCenter 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. BillingCenter imports no data into the server. Use this option to convert CSV input files to XML-formatted output files.
`-password` *password*	Specifies the *password* to use to connect to the server. BillingCenter requires the password.
`-privileges`	Rebuilds role privileges by deleting role privileges in the current database and importing the `roleprivileges.csv` file in `BillingCenter/modules/configuration/config/import/gen`.
`-server` *url*	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-user` *user*	The user (*user*) to use to run this process.

Maintenance Tools Command

```
maintenance_tools -help
maintenance_tools -password password [-server url] [-user user] {
                  -importdefaultsampledata |
                  -processstatus process |
                  -startprocess process |
                  -terminateprocess process }
```

The `maintenance_tools` command starts, terminates, or retrieves the status of a BillingCenter process.

See also

- For a list of processes that the `maintenance_tools` command can start, see "Scheduling Work Queue Writers and Batch Processes" on page 113.
- "Maintenance Tools Web Service" on page 128 in the *Integration Guide*

Maintenance Tools Options

You can use any of the following options with the `maintenance_tools` command. You must always supply the `-password` option.

Option	Description
`-importdefaultsampledata`	Imports the BillingCenter default sample data set.
`-password` *password*	Specifies the administrative password. BillingCenter requires a password to launch the maintenance tools.
`-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 there are remaining work items. It is possible for the process status to report as completed as the writer has completed adding items to the work queue, yet, there are remaining unprocessed work items.
`-server` *url*	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-startprocess` *process*	Starts a new batch process. For the *process* value, specify a valid process code. For a list of batch process codes, including work queue writer processes, see "Scheduling Work Queue Writers and Batch Processes" on page 113
`-terminateprocess` *process*	Terminates a batch process. For the *process* value, specify a valid process name or a process ID. 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 "List of Work Queues and Batch Processes" on page 120 to determine if it is possible to terminate a process.
`-user` *user*	The user (*user*) to use to run this process.

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 |
                -retry messageID |
                -retrydest destinationID |
                -skip messageID |
                -statistics destinationID |
                -suspend destinationID }
```

The `messaging_tools` command lets you manage a message destination from the command prompt. To manage a message destination from the command prompt, you must know its destination ID. The person who creates the message destination assigns this ID.

Messaging Tools Options

You can use any of the following options with the `messaging_tools` command. You must always supply the `-password` option.

Option	Description
`-config -destination` *`destinationID`*	Returns the configuration for a message destination.
`-destination` *`destinationID`*	Specifies a message destination (*`destinationID`*).
`-password` *`password`*	Specifies the administrative password. You must specify a *`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.
`-restart -destination` *`destinationID`* `-wait` *`wait`* `-retries` *`retries`* `-initial` *`initial`* `-backoff` *`backoff`* `-poll` *`poll`* `-threads` *`threads`* `-chunk` *`chunk`*	Restarts the messaging destination with new configuration settings: • *`destination`* – The destination ID of the destination to 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, BillingCenter 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 "Message Ordering and Multi-Threaded Sending" on page 337 in the *Integration Guide* for details on how the polling interval works. If your performance issues primarily relate to many messages per primary object per destination, then the polling interval is the most important messaging performance setting. • *`threads`* – To send messages associated with a primary object, BillingCenter 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. This setting is ignored for non-safe-ordered messages, since those are always handled by one thread for each destination. If your performance issues primarily relate to many messages but few messages per claim for each destination, then this is the most important messaging performance setting. For more information, see "Message Ordering and Multi-Threaded Sending" on page 337 in the *Integration Guide*. • *`chunk`* – number of messages to read in a chunk.
`-resume -destination` *`destinationID`*	Resumes the operation of the specified message destination.
`-retry` *`messageID`*	Attempts to resend a message that failed. The message must be a candidate for retrying. A message is a candidate if the error at the destination system was temporary and the message destination has no automatic retry mechanism. For instance, if the record was locked and refused the update, the message would be a candidate for retrying.

Option	Description
`-retrydest destinationID`	Retries all retryable messages for a message destination.
`-server url`	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-skip messageID`	Skips a message with the specified ID. If you mark a message as skipped, then BillingCenter 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 BillingCenter processes a daily batch file.
`-user user`	The user (*user*) to use to run this process.

System Tools Command

```
system_tools -help
system_tools -password password [-server url] [-user user] {
             -cancelupdatestats |
             -checkdbconsistency [-tableselection tblSelection -checkTypeSelction checkTypeSelection] |
             -daemons |
             -dbcatstats [regularTables stagingTables typelistTables] |
             -getdbstatisticsstatements |
             -getincrementaldbstatisticsstatements |
             -getPerfReport ID |
             -getupdatestatsstate |
             -listPerfReports [number] |
             -loggercats |
             -maintenance |
             -mssqlPerfRpt [numTopQueries numHotObjects collectStatistics] |
             -multiuser |
             -oraListSnaps numstats |
             -oraPerfReport beginSnapshotID endSnapshotID probeVDollarTables |
             -ping |
             -recalcchecksums |
             -reloadloggingconfig |
             -sessioninfo |
             -updatelogginglevel loggername logginglevel |
             -updatestatistics description update |
             -verifydbschema |
             -version }
```

See also

- “System Tools Web Services” on page 129 in the *Integration Guide*

System Tools Options

You can use any of the following options with the `system_tools` command. You must always supply the `-password` option.

Option	Description
`-cancelupdatestats`	Cancels the process that is updating database statistics 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. BillingCenter provides this option so that you can schedule consistency checks to run when the database server is handling fewer requests. 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. • *table name* – The name of a single table on which to run checks. • `tg.`*table group name* – The name of a table group. Table groups are defined in the `database` element of `config.xml`. For more information, see "Defining Table Groups" on page 65 in the *Installation Guide*. • `@`*file name* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with `tg.`, such as `tg.MyTableGroup`. You can combine table groups and individual table names in the same file. Specify the *checkTypeSelection* argument as one of the following: • `all` – Run all consistency checks on the specified tables. • *check name* – The typecode of a single consistency check to run. • `@`*file name* – A file name with one or more valid consistency check names entered in comma-separated values (CSV) format. If you specify one optional argument, you must specify both. To run consistency checks from BillingCenter, use the **Server Tools** → **Consistency Checks** → **Info Page** screen, described in "Consistency Checks" on page 161. For more information, see "Checking Database Consistency" on page 36.
`-daemons`	Sets the server to the `DAEMONS` run level. For information about functionality available at various run levels, see "Server Modes and Run Levels" on page 56.

Option	Description
`-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/none tables of this type, or • <*table name*> – The name of a single table of this type, or • @<*file name*> – A file name with one or more valid table names of this type entered in comma-separated values (CSV) format. For example, `-dbcatstats none none all` 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, BillingCenter 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 “Configuring Database Statistics” on page 39.
`-getdbstatisticsstatements`	Retrieves the list of SQL statements to update database statistics and prints the list to the console. See “Configuring Database Statistics” on page 39.
`-getincrementaldbstatisticsstatements`	Retrieves the list of SQL statements to update database statistics for tables exceeding the change threshold. Prints the list to the console. The change threshold is defined by the `incrementalupdatethresholdpercent` attribute of the `databasestatistics` element in `database-config.xml`. See “Configuring Database Statistics” on page 39.
`-getPerfReport` *ID*	Downloads the performance report with the specified *ID*. You can retrieve a list of available performance report IDs by running the `-listPerfReports` command option.
`-getupdatestatsstate`	Returns the state of the process running the statistics update.
`-listPerfReports` *number*	Lists IDs and other information for available database performance reports. You can specify an optional integer (*number*) to specify the number of available downloads to list, ordered starting with the most recent. If unspecified or `0`, all available downloads are listed. The list shows the ID of the report and the status, indicating if the performance report batch job succeeded, failed, or is still running. The list also includes the start and end times of the batch job and the description of the batch run. You can use the ID of the performance report to download the report with the `-getPerfReport` *ID* option.
`-loggercats`	Displays the available logging categories.
`-maintenance`	Sets the server to the `MAINTENANCE` run level. For information about functionality available at various run levels, see “Server Modes and Run Levels” on page 56.

Option	Description
`-mssqlPerfRpt` *`numTopQueries numHotObjects collectStatistics`*	Generates a SQL Server DMV (Dynamic Management Views) performance report using the `MSDMReport` batch process. This command option has the following arguments: • *`numTopQueries`* • *`numHotObjects`* • *`collectStatistics`* Replace *`numTopQueries`* and *`numHotObjects`* with integer values for the number of top queries and hot objects to report. Replace *`collectStatistics`* with `true` or `false` to specify whether BillingCenter gathers database statistics while generating the DMV report. You must specify all three arguments or none. If you do not specify any arguments, BillingCenter uses defaults of 400 top queries, 400 hot objects, and collects statistics.
`-multiuser`	Sets the server to the `MULTIUSER` run level. For information about functionality available at various run levels, see “Server Modes and Run Levels” on page 56.
`-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 an Oracle 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. The third argument can also specify a file by prefixing the filename with an `@` sign, for example, `@`*`filename`*`.properties` Optionally, you can prefix the filename 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 BillingCenter 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, BillingCenter 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 `-processstatus` *`processID`* option. View the performance report on the Info Page. See “Oracle AWR” on page 169.
`-password` *`password`*	Specifies the administrative password. You must specify a *`password`*.

Option	Description
`-ping`	Pings the server to check if its active. The returned message indicates the server run level. The possible responses are: • `MULTIUSER` • `DAEMONS` • `MAINTENANCE` • `STARTING` For information about functionality available at various run levels, see "Server Modes and Run Levels" on page 56.
`-recalcchecksums`	Recalculates file checksums used for clustered configuration verification.
`-reloadloggingconfig`	Directs the server to reload the logging configuration file.
`-server` *`url`*	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-sessioninfo`	Returns the session information of the server.
`-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 a process to update database statistics. Specify a value of `true` for *`update`* to update database statistics only for tables exceeding the change threshold. The change threshold is defined by the `incrementalupdatethresholdpercent` attribute of the `databasestatistics` element in `database-config.xml`. Specify `false` to gather full database statistics. The description is shown on the **Execution History** tab of the **Database Statistics** info page. See "Configuring Database Statistics" on page 39.
`-user` *`user`*	The user (*`user`*) to use to run this process.
`-verifydbschema`	Verifies that the data model matches the underlying physical database.
`-version`	Returns the running server version, the database schema version, and configuration version.

Table Import Command

```
table_import -help
table_import -password password [-server url] [-user user] {
             -clearerror |
             -clearexclusion |
             -clearstaging
             -deleteexcluded [-batch] |
             -encryptstagingtbls [-batch] |
             -getLoadHistoryReport reportID [-filepath filepath] |
             -integritycheck [-allreferencesallowed] [-batch] [-clearerror] [-populateexclusion] |
             -integritycheckandload [allreferencesallowed] [-batch] [-clearerror] [-estimateorastats]
                 [-populateexclusion] [-zonedatanly] |
             -listLoadHistoryReports numReports |
             -populatexclusion [-batch] |
             -updatedatebasestatistics [-batch] [-integritycheckandload] }
```

The `table_import` command loads data from staging tables into BillingCenter. Before you can use this command, use the `system_tools -daemons` command option to set the server run level to `MAINTENANCE`.

It is not possible to use the `system_tools -terminateprocess` command option to terminate the `table_import` command.

See also

- "System Tools Command" on page 191.

- For more information on database staging tables, see “Importing from Database Staging Tables” on page 367 in the *Integration Guide*.
- For information on the web service `TableImportAPI` that also loads data from staging tables, see “Table Import Tools” on page 377 in the *Integration Guide*.

Table Import Options

You can use any of the following options with the `table_import` command. You must always supply the `-password` option.

Option	Description
`-allreferencesallowed`	Allows references to existing rows in all source tables, including administrative tables such as users and groups.

`-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`

`-clearerror`	Clears the error table.
`-clearexclusion`	Clears the exclusion table.
`-clearstaging`	Clear the staging tables.
`-deleteexcluded`	Deletes rows from staging tables based on contents of exclusion table.
`-encryptstagingtbls`	Instructs BillingCenter to encrypt the columns that are marked for encryption in staging tables using the current encryption plugin. See “Encryption Features for Staging Tables” on page 259 in the *Integration Guide*.
`-estimateorastats`	Updates database statistics on the source tables with estimated row and block counts for the source tables and indexes at the beginning of load (`-integritycheckandload`). This command option applies only to Oracle databases.
-filepath *`filepath`*	Path to target directory in which to download a report.
-getLoadHistoryReport *`reportID`*	Downloads a zipped version of the load history report as specified by the value of *`reportID`*. (Use the `-listLoadHistoryReports` option to determine the ID to use.) Use the optional `-filepath` parameter to specify the target directory for the download.

`-integritycheck`	Validates the contents of the staging tables. You can optionally specify:
	`-allreferencesallowed`
	`-clearerror`
	`-populateexclusion`

`-integritycheckandload`	Validates the contents of the staging tables and populate source tables. You can optionally specify one of the following command options as well:
	`-allreferencesallowed`
	`-clearerror`
	`-estimateorastats`
	`-populateexclusion`
	`-zonedataonly`

`-listLoadHistoryReports` [*`numReports`*]	Lists the most recent load history reports. Optional parameter *`n`* is the number of reports to list: • If you supply a positive integer for *`numReports`*, then BillingCenter lists that number of most recent reports. • If you do not supply a value for *`numReports`*, then BillingCenter lists all available reports.
`-messagesinks` *`sinks, ...`*	Deprecated. This option does not do anything.
`-password` *`password`*	Specifies the administrative password. You must specify a *`password`*.
`-populateexclusion`	Populate the exclusion table with rows to exclude.

Option	Description
`-server url`	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-updatedatabasestatistics`	Updates the database statistics on the staging tables. If you also specify the `-integritycheckandload` command option, the `-updatedatabasestatistics` option calculates the estimated row and block counts for the source tables. If running against an Oracle database, the `-updatedatabasestatistics` option updates indexes before populating the staging tables.
`-user user`	The user (*user*) to use to run this process.
`-zonedataonly`	Sets the import to load zone data only. Used with the `-integritycheckandload` command option.

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 `WorkflowAPI`. See “Workflow Web Services” on page 130 in the *Integration Guide*.

Workflow Tools Options

You can use any of the following options with the `workflow_tools` command. You must always supply the `-password` option.

`-complete workflowID`	Completes running workflow for the specified workflow (*workflowID*).
`-password password`	Specifies the administrative password. You must specify a *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 BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-suspend workflowID`	Suspend the named workflow (*workflowID*).
`-user user`	The 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 BillingCenter, 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 "Importing Zone Data" on page 105.
- For more information on database staging tables, see "Importing from Database Staging Tables" on page 367 in the *Integration Guide*.
- For information on the web service `ZoneImportAPI` that also imports zone data, see "Introduction to Database Staging Table Import" on page 367 in the *Integration Guide*.

Zone Import Options

You can use any of the following options with the `zone_import` command. You must always supply the `-password` option.

Option	Description
`-charset` *charset*	Character set encoding of the zone data file. The default is UTF-8.
`-clearproduction`	Clears zone data from the production tables. Optionally, specify the `-country` option to clear data for only one country.
`-clearstaging`	Clears zone data from the staging tables. Optionally, specify the `-country` option to clear data for only one country.
`-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, BillingCenter clears the data in the staging tables for the specified country before importing the data from the import file.
`-password` *password*	Specifies the administrative password. You must specify a password.
`-server` *url*	Specifies the BillingCenter host server URL. Include the port number and web application name, for example: `http://servername:8580/bc`
`-user` *user*	The user (*user*) to use to run this process.