ClaimCenter System Administration Guide

Release 6.0.8

Product Name: Guidewire ClaimCenter
Product Release: 6.0.8
Document Name: *ClaimCenter System Administration Guide*
Document Revision: 05-February-2013

Contents

About This Document

This document provides guidance for the management of a ClaimCenter system.

This topic includes:

Intended Audience

This document is intended to help system administrators monitor ClaimCenter, manage its security, and take care of routine tasks, such as system back-ups, logging, and importing data.

Assumed Knowledge

This document assumes that you are already familiar with system administration for your database server, application server, and operating system. It addresses only administration of the ClaimCenter application itself.

Related Documents

See the following Guidewire documents for further information:

ClaimCenter Installation Guide – Describes how to install a new copy of ClaimCenter into Windows or UNIX environments. This guide is intended for system administrators and developers who need to install ClaimCenter.

ClaimCenter Upgrade Guide – Provides instructions to upgrade ClaimCenter.

ClaimCenter Configuration Guide – Describes how to configure ClaimCenter and includes basic steps and examples for implementing such configurations. This guide is intended for IT staff and system integrators who configure ClaimCenter for an initial implementation or create custom enhancements. This guide is intended as a reference, not to be read cover-to-cover.

ClaimCenter Integration Guide – Provides an architectural overview and examples of how to integrate ClaimCenter with external systems and custom code. This document is a learning tool for explanations and examples with links to the *Java API Reference Javadoc* and *SOAP API Javadoc* for further details. This document is written for integration programmers and consultants.

ClaimCenter Rules Guide – Describes the business rule methodology, rule categories for ClaimCenter, and rule syntax for Guidewire Studio. This book is intended for programmers who write Gosu business rules and analysts who define the business rule logic.

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.	Get the field from the `Address` object.
`monospaced italic`	Parameter names or other variable placeholder text within URLs or other code snippets.	Use `getName(`*`first`*`, `*`last`*`)`. `http://`*`SERVERNAME`*`/a.html`.

Support

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

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

chapter 1

Installation Files and Directories

This topic discusses the important files and directories that make up the ClaimCenter installation. Additionally, this topic provides an overview of the administrative commands provided with ClaimCenter and information about when to regenerate and redeploy ClaimCenter.

This topic includes:

The ClaimCenter Installation Directory

The ClaimCenter installation directory contains the following directories:

Directory	Description
`admin`	Contains administrative tools. See "ClaimCenter Administrative Commands" on page 169 for descriptions.
`bin`	Contains the `gwcc` batch file and shell script used to launch commands for building and deploying. See "Commands Reference" on page 69 in the *Installation Guide*.
`build`	Contains products of build commands such as exploded WAR and EAR files and the data and security dictionaries. This directory is not present when you first install ClaimCenter. The directory is created when you run one of the build commands.
`dist`	Guidewire application EAR, WAR, and JAR files are built in this directory. The directory is created when you run one of the build commands to generate EAR or WAR files.
`doc`	HTML and PDFs of ClaimCenter documentation.
`java-api`	Contains the Java API libraries and Javadoc documentation. You create these by running the `gwcc regen-java-api` command. See "Regenerating the Integration Libraries" on page 17 in the *Integration Guide*.
`modules`	Contains subdirectories including configuration resources for each application component.
`soap-api`	Contains the SOAP API libraries and Javadoc documentation. You create these by running the `gwcc regen-soap-api` command. See "Regenerating the Integration Libraries" on page 17 in the *Integration Guide*.

studio	Contains Studio preferences and TypeInfo database caches. Studio generates this directory when you first launch Studio.
webapps	Contains necessary files for using QuickStart or Tomcat application servers or WebLogic and WebSphere application servers for development.

How ClaimCenter Interprets Modules

ClaimCenter groups configuration resources in module directories within ClaimCenter/modules. ClaimCenter evaluates these resources at startup according to a specific order. ClaimCenter always checks first for a resource in the configuration module.

The module directories might contain distinct copies of the same resource file. In that case, the predominant copy is the first one ClaimCenter finds. ClaimCenter disregards any others.

For example, in the base install, the file Desktop.pcf resides in the following directory:

ClaimCenter/modules/cc/config/web/pcf/desktop

If you edit this file from Guidewire Studio, Studio places a new copy in the configuration directory. The original file remains, but ClaimCenter ignores it for as long as the edited one exists. If you delete the edited copy from the configuration module, ClaimCenter uses the copy in the cc module.

Edited Resource Files Reside ONLY in Configuration Module

The configuration module is the only place for configured resources. As ClaimCenter starts, a checksum process verifies that no files have been changed in any directory except for those in the configuration directory. If this process detects an invalid checksum, ClaimCenter does not start. In this case, overwrite any changes to all modules except for the configuration directory and try again.

If you use Guidewire Studio to edit a configuration file, Studio automatically copies the file to the configuration module, if a copy is not already present. Guidewire recommends that you use Studio to edit configuration files to minimize the risk of accidentally editing a file outside the configuration module.

Key Directories

The installation process creates a configuration environment for ClaimCenter. In this environment, you can find all of the files needed to configure ClaimCenter in two directories:

- The main directory of the configuration environment. In the default ClaimCenter installation, the location of this directory is ClaimCenter/modules/configuration.
- ClaimCenter/modules/configuration/config contains the application server configuration files.

The installation process also installs a set of system administration tools in ClaimCenter/admin/bin.

ClaimCenter runs within a J2EE server container. To deploy ClaimCenter, you build an application file suitable for your server and place the file in the server's deployment directory. The type of application file and the deployment directory location is specific to the application server type. For example, for ClaimCenter (deployed as the cc.war application) running on a Tomcat J2EE server on Windows, the deployment directory might be C:\Tomcat\webapps\cc.

For instructions on building and deploying ClaimCenter, see the *ClaimCenter Installation Guide*.

Build Tools

Guidewire provides several commands that you can use to build and deploy ClaimCenter. These commands are launched using the `gwcc` utility in `ClaimCenter/bin`.

Command	Action
`gwcc -p`	Displays all `gwcc` command options.
`gwcc build-war`	Builds the generic WAR file for Tomcat. You can include the Boolean parameter `config.war.dictionary=true` to also generate the ClaimCenter *Data Dictionary* and *Security Dictionary* while building the WAR file. Use the following command: `gwcc build-war -Dconfig.war.dictionary=true` When `config.war.dictionary=true`, the command creates a `dictionary` folder within the WAR file. The `dictionary` folder contains `data` and `security` folders. These folders contain the *Data Dictionary* and *Security Dictionary* respectively. To view a dictionary, open `index.html` in the `data` or `security` folder.
`gwcc build-weblogic-ear`	Builds the EAR file for WebLogic.
`gwcc build-websphere-ear`	Builds the EAR file for WebSphere.
`gwcc copy-starter-resources`	Copies starter configuration resources to the configuration module, if applicable. Class: `com.guidewire.tools.config.CopyStarterConfigResourcesTool`
`gwcc regen-datamapping-split`	Builds the data mapping files with files split out by table and typelist. Class: `com.guidewire.tools.datamapping.DataMappingTool`
`gwcc regen-datamapping-together`	Builds the data mapping files with all tables and typelists concatenated. Class: `com.guidewire.tools.datamapping.DataMappingTool`
`gwcc regen-dictionary`	Generates the *ClaimCenter Data Dictionary* and *ClaimCenter Security Dictionary*. Generate the first time you unzip the application and each time you update the data model. Run the `gwcc regen-java-api` and `gwcc regen-soap-api` commands each time just prior to regenerating the security and data dictionaries. To view either, open a new browser window and enter: • `ClaimCenter/dictionary/data/index.html` for the *Data Dictionary* or • `ClaimCenter/dictionary/security/index.html` for the *Security Dictionary*. You can also generate these dictionaries while building a WAR file. See the description for `gwcc build-war` for instructions. Classes: `com.guidewire.tools.dictionary.data.DataDictionaryTool` `com.guidewire.tools.dictionary.security.SecurityDictionaryTool`
`gwcc regen-gosudoc`	Generates Gosu documentation similar to JavaDoc but using the ClaimCenter type system. This command produces documentation at `ClaimCenter/build/gosudoc/index.html`. See “Gosu Generated Documentation” on page 35 in the *Gosu Reference Guide*. Class: `com.guidewire.tools.gosudoc.GosuDocMain`
`gwcc regen-java-api`	Builds the Java API toolkit and examples to the `ClaimCenter/java-api` directory. See “Regenerating the Integration Libraries” on page 17 in the *Integration Guide*. Class: `com.guidewire.commons.entity.external.ExternalGenerator`
`gwcc regen-pcfmapping`	Builds the PCF mappings. Class: `com.guidewire.tools.pcfmapping.PCFMappingWriterMain`
`gwcc regen-rulereport`	Generates an XML report describing the existing business rules. See “Generating a Rule Repository Report” on page 91 in the *Rules Guide*.

Command	Action
`gwcc regen-soap-api`	Builds the SOAP API toolkit and examples to the `ClaimCenter/soap-api` directory. See “Regenerating the Integration Libraries” on page 17 in the *Integration Guide*. Classes: `com.guidewire.tools.wsdl.WSDLGenerator` `com.guidewire.util.webservices.axis.WSDLToJavaGenerator`
`gwcc regen-xsd`	Builds the XSD files for data import. See “Creating an XML File for Import” on page 121 in the *System Administration Guide* and “Importing Administrative Data” on page 83 in the *Integration Guide*.
`gwcc studio`	Runs Guidewire Studio. Class: `com.guidewire.studio.main.Main`
`gwcc verify-checksum`	Verifies module checksums. Class: `com.guidewire.tools.checksum.ModulesChecksumTool`
`gwcc verify-types`	Checks PCF files for errors. See“Validating Studio Resources” on page 121 in the *Configuration Guide*.
`gwcc version`	Displays the product version.

Administration Tools

ClaimCenter provides a set of administrative tools you can use to control the server from the command line. Typically, these commands are meant to run on an administrator’s workstation. The tools are all found in the `ClaimCenter/admin/bin` directory.

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 ClaimCenter server is running.

Unless otherwise specified, details of each command are located in “ClaimCenter Administrative Commands” on page 169.

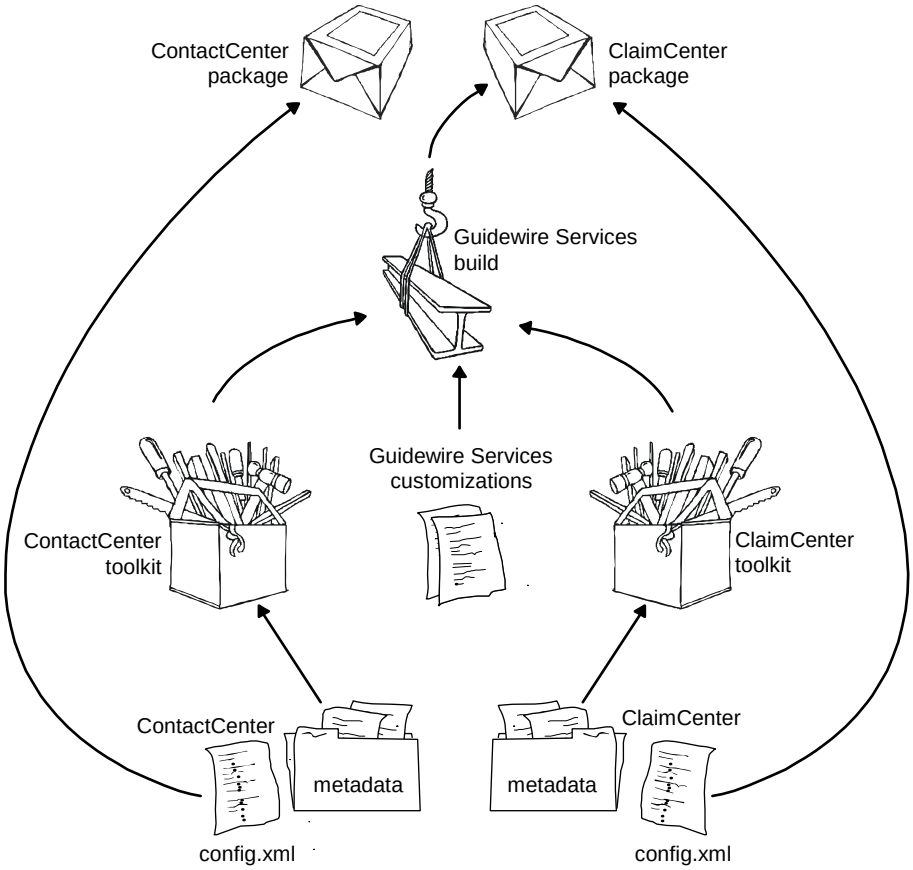
The following table summarizes what each tool does.

File	Description
`fnol_import`	Imports or exports First Notice Of Loss (FNOL) data.
`fnol_mapper`	A ClaimCenter integration tool that imports FNOL reports, which are initial claim reports, from a standard XML-based file format called ACORD XML. See “FNOL Mapper” on page 381 in the *Integration Guide* for details.
`import_tools`	A set of utilities for loading XML-formatted data into ClaimCenter.
`maintenance_tools`	A 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.
`template_tools`	Helps in converting between template versions.
`usage_tools`	Provides data about the exposure data in the system.
`workflow_tools`	Allows you to manage user workflows in the system.
`zone_import`	Loads zone data from a file to a staging table.

Knowing When to Regenerate and Redeploy

The process for deploying configuration changes is similar to that of the initial installation. The difference is that deploying changes overwrites the old files. Back up the old WAR or EAR file prior to deploying configuration changes. For instructions, see “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

To understand when you need to regenerate the toolkit and redeploy the application package, it is important to understand the dependencies. If your environment includes multiple applications integrated through Guidewire Services, the dependencies are more complex. The following figure illustrates the dependencies between two Guidewire applications in such an environment:



The table describes the dependencies illustrated by the diagram:

Target	Dependencies
Integrated Guidewire Applications	ClaimCenter toolkit, ContactCenter toolkit, Guidewire Services build customizations (for example, `cc-to-ab-data-mapping.xml`).
ClaimCenter application package	Integration build, ClaimCenter configuration (including ClaimCenter metadata).
ClaimCenter toolkit	ClaimCenter metadata (extensions, typelists).
ContactCenter application package	ContactCenter configuration (including the ContactCenter metadata).
ContactCenter toolkit	ContactCenter metadata (extensions, typelists).

In a production environment, Guidewire recommends that you always deploy configuration changes by rebuilding the WAR file. In a development or test environment, you can choose to rebuild the WAR file or not. For most changes to the `config.xml` file, you need only copy the configuration to the deployment directory. The

ClaimCenter toolkit depends on the ClaimCenter extensions and typelists. If you change either of these, you need to regenerate the toolkit. If you regenerate the toolkit, rebuild the application package, which is also dependent on this data.

chapter 2

Basic Configuration

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

For information about the ClaimCenter installation directory contents, see “ClaimCenter Configuration Files” on page 88 in the *Configuration Guide*.

This topic includes:

The Application Server config.xml

ClaimCenter provides many system parameters to configure its behavior. You set these parameters in the `ClaimCenter/modules/configuration/config/config.xml` file. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**. The `config.xml` file includes database connection settings and ClaimCenter 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.

The `config.xml` file also includes a few other database settings that you might want to change as part of ongoing system administration, including automatic database updates and database consistency checking.

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

Defining the Application Server Environment

During startup, ClaimCenter 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 ClaimCenter 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, ClaimCenter sets it to the `hostname` of the ClaimCenter server. Log entries display only the first 10 characters of the `serverid` value.
`env`	The name of the environment in which the server operates. The default is `null`.
`isbatchserver`	Whether the server is a batch server or not. In a clustered environment, the default value is `false`. In a non-clustered environment, the batch server resolves to `true`.

You can specify environment property values through JVM options or through the `registry` element. Pass JVM options through the command line you use to start the application server. Define the `registry` element in the `config.xml` file. Depending on how many ClaimCenter 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.

Notes

- If you intend to use clustering in your environment, read this section thoroughly before going on to read "Managing Clustered Servers" on page 81. 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 the *ClaimCenter Rules Guide* for information on using this library.

Setting Java Virtual Machine (JVM) Options

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

- `gw.cc.serverid`
- `gw.cc.env`
- `gw.cc.isbatchserver`

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

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

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

- 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.

Using the registry Element to Specify Environment Properties

As an alternative to using JVM options, you can specify environment properties 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.cc.*` Java system properties and set default values. This element has the following attributes:

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

This value overrides the name of a default `gw.cc.*` 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.cc.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, ClaimCenter calculates environment properties. This section describes how ClaimCenter calculates each environment property.

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

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

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

env Property

If you specify the -Dgw.cc.env JVM option, ClaimCenter sets env to that value. Alternatively, you can define the option name using the env environment property and set a default value. The following example shows how to set the env using 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, ClaimCenter sets the env to the test value. If you do not specify the option, ClaimCenter 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, ClaimCenter sets the env to null.

Note: The env property has special significance for logging behavior. If the env value is non-null, ClaimCenter 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 ClaimCenter 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 85.

For more information on batch processes, and work queues that distribute batch processing across multiple servers, see “Batch Processes and Work Queues” on page 129.

serverid Property

If you specify the -Dgw.cc.serverid JVM option, ClaimCenter sets the serverid to that value. Alternatively, you can define the serverid value using the serverid environment property in the registry element, as shown in the following example:

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

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

</registry>
```

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

1. If you specify the JVM option -Dgw.cc.serverid (or the value of the serverid property defined in a <systemproperty>), ClaimCenter uses that value for the serverid. For example, while starting the application server, include -Dgw.cc.serverid=prod1 to set the serverid to prod1.
2. If you do not specify the JVM option, ClaimCenter 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, ClaimCenter sets serverid to the host name of the computer. Under some extreme security settings this is not

available, in which case ClaimCenter sets the `serverid` to `localhost`. If you run multiple servers on the same host computer, define a `serverid` for each server using 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, ClaimCenter 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 ClaimCenter 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.cc.env=test
```

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

If you define environment-specific parameters, ClaimCenter 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"/>
```

ClaimCenter sets `BusinessDayStart` to `9:00 AM` if either `env` resolves to `test` or `serverid` resolves to `prodserver`. For example, if ClaimCenter resolves `env` to `test` and `serverid` to `chicago`, the `BusinessDayStart` is `9:00 AM`. Similarly, if ClaimCenter 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`, ClaimCenter uses the default `BusinessDayStart` of `8:00 AM`.

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

Providing Default Values

At startup, ClaimCenter 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 ensure that 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 are working in the test or development environment. To do this, define two database elements in `config.xml`, as shown in the following example:

```
<database name="ClaimCenterDatabase"
  driver="dbcp"
```

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

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

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

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

Using the Geocoding Feature

Guidewire supports *geocoding* within ClaimCenter and in ContactManager. Geocoding is the ability to assign latitude and longitude to an address. Software can then use these codes or coordinates to present a user with, for example, the distance between two addresses. Within both ClaimCenter and ContactCenter, all primary addresses are candidates for geocoding.

Working with the Geocode Plugin

To implement geocoding features, use a `GeocodePlugin` implementation for a specific external geocoding service to provide latitude and longitude coordinates for specific addresses. Typically, plugin implementations also use an external mapping service to calculate and return proximity information, driving instructions, and maps.

Guidewire provides you with a fully functioning and supported `GeocodePlugin` implementation. This implementation connects to the Microsoft Bing Maps Geocode Service. If you intend to use this plugin, your organization must have a valid account with Microsoft. This Bing Maps plugin is written in Gosu. Guidewire Studio lets you enable the plugin, specify which implementation to use, and specify parameters for the implementation.

For more information about implementing a geocode plugin, see “Geographic Data Integration” on page 315 in the *Integration Guide*.

IMPORTANT Geocoding a large number of addresses is resource intensive. Geocode existing addresses during periods of minimal activity on the ClaimCenter server.

The `Geocode` batch process regularly searches for new addresses as specified in the `scheduler-config.xml` file. See “Scheduling Geocoding” on page 24 to configure this file. You can also run the process manually from the command line. See “Running Batch Processes from the Command Line” on page 147 for instructions.

The `Geocode` work queue has the potential to alter a large number of `Address` records. Update database statistics after running the `Geocode` work queue by running the `incrementaldbstats` process. See “Configuring Database Statistics” on page 53.

Understanding Address Processing

The geocoding feature uses the Guidewire work queue infrastructure. This infrastructure enables asynchronous processing of addresses stored in the database. The work queue infrastructure supports both a daemon writer and worker. The writer daemon searches the database for addresses without geocodes and marks them for geocoding. The worker daemon submits the marked addresses to the `GeocodePlugin`. After the plugin returns geocode coordinates for an address, the worker daemon retrieves them and saves the updated address to the database. If an address changes within the submission and retrieval interval, the worker daemon resubmits it.

The work queue infrastructure is especially useful in a clustered environment as it enables parallel processing of addresses across the cluster members. In a clustered environment, the worker and writer daemon can run on any server in a cluster, not just the batch server.

> **Note:** When you first start ClaimCenter, there might be a lot of addresses to geocode, particularly if you imported a large number of new addresses into a production database. In such a situation, the `GeocodePlugin` might take a lot of time to process these new addresses.

Configuring Geocoding

Configuration involves enabling the `GeocodePlugin` and setting its configuration parameters in both ClaimCenter and ContactCenter. By default, the `GeocodePlugin` is disabled but configured to use the Bing Maps `GeocodePlugin` implementation.

> **Note:** To use this plugin, your company must have its own account, login, and application key with Bing Maps. For more information, go to `http://www.bingmapsportal.com`, where you can set up a Bing Maps account and obtain an application key. There is also a video tutorial at that site describing how to get a key. When you create a key, the application name is arbitrary and no application URL is required.

To enable and use the plugin in ClaimCenter

1. Start ClaimCenter Studio.

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

   ```
   gwcc studio
   ```

2. In the **Resources** pane, navigate to **configuration** → **Plugins** → **gw** → **plugin** → **geocode** → **GeocodePlugin**.
3. Click the **Enabled** check box to enable the plugin. If you see a message asking if you want to create a copy in the current module, click **Yes**.
4. Make sure that the **Class** field specifies the Bing Maps implementation class:

   `gw.plugin.geocode.impl.BingMapsPlugin`
5. Under **Parameters**, click the **Value** field for **applicationKey**, and then enter the application key you obtained from Bing Maps.
6. Save your changes.

> **IMPORTANT** If you use ContactCenter, repeat your changes in ContactCenter Studio.

Geocoding Parameters

Configure geocoding with the following parameters in `config.xml`:

Parameter	Description	default
`UseGeocodingInPrimaryApp`	For ClaimCenter, enables geocoding and proximity search for users in assignment user interface. For ContactCenter, does nothing.	`false`

Parameter	Description	default
UseGeocodingInAddressBook	Set to true if you have ClaimCenter integrated with ContactCenter and ContactCenter has geocoding enabled for vendors. This setting enables vendor search in the ClaimCenter and ContactCenter user interfaces.	false
UseMetricDistancesByDefault	Use kilometers instead of miles in driving directions. Set this parameter identically in both ClaimCenter and ContactCenter.	false
ProximitySearchOrdinalMaxDistance	Maximum distance to use while performing an ordinal (nearest N) proximity search. This distance is in miles unless UseMetricDistancesByDefault is true. You **must** set this parameter identically in both ClaimCenter and ContactCenter.	300

Configure the Geocode distributed work queue by modifying the following section in work-queue.xml:

```
<work-queue name="Geocode" progressinterval="600000">
    <worker instances="1" batchsize="100"/>
</work-queue>
```

The default configuration specifies one Geocode worker to pass addresses found by the writer daemon to the geocode application and update the database with the results.

For more information about distributed work queues, see “Batch Processes and Work Queues” on page 129.

Scheduling Geocoding

You can configure the schedule for running the Geocode writer process by modifying the scheduler-config.xml file. By default, the scheduler for the Geocode writer is disabled. To enable the scheduler for the Geocode writer, uncomment the following section:

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

The Geocode writer process is set to run at 1:30 AM daily. You can configure the schedule. See “Scheduling Batch Processes and Distributed Work Queues” on page 143 for instructions.

If adding many new contacts, especially into ContactCenter, tune the scheduler to match the expected daily load of new addresses.

Geocoding and Multiple Guidewire Products

Configuring the geocoding plugin within ClaimCenter enables you to geocode user addresses stored in the ClaimCenter database *only*. If you integrated ClaimCenter with ContactCenter, an **AddressBook** tab appears in ClaimCenter. To make the addresses in the **AddressBook** tab available for geocoding, you must do the following:

- Use ContactCenter’s Studio to define the ContactCenter geocoding plugin.
- In ContactCenter, specify the daemon worker and writer configuration parameters using the <work-queue> elements.
- In ContactCenter, set the geocoding parameters as described above in “Geocoding Parameters” on page 23.

Geocode Status

The GeocodeStatus type list defines the set of status codes returned from the plugin. This type list is final and cannot be edited. Access the GeocodeStatus typelist from the Guidewire Studio **Resources** pane under **configuration** → **Typelists** → **GeocodeStatus.** See also “Geocoding Status Codes” on page 323 in the *Integration Guide*.

Configuring Guidewire Document Assistant

ClaimCenter provides the `TemplateRunner.ScriptControl` ActiveX control to perform many of the functions related to document manipulation. ClaimCenter attempts to install this control as a client initiates a session. The browser prompts the user to install the Guidewire Document Assistant. This is synonymous with the `TemplateRunner.ScriptControl` ActiveX control.

WARNING The Guidewire Document Assistant, like all ActiveX controls, has known security vulnerabilities. Contact Guidewire support for more information if you plan to use the Guidewire Document Assistant ActiveX control in production.

For more information about document management, see “Document Management” on page 249 in the *Integration Guide*.

This topic includes:

What the Control Provides ClaimCenter

This control provides ClaimCenter with the following features:

- Support for the automatic creation of new documents from custom document templates.
- Provide a more secure means of supplying a local client-side path name to the server during a file upload.
- Support client-side JScript required for some templates, for example the MailMerge templates. (The MailMerge template is a Guidewire-supplied reference implementation.) For this purpose, `TemplateRunner.ScriptControl` creates a temporary file containing the required JScript and then runs it using the Windows Script Host. To support this mechanism, the `.js` extension on the user’s environment must map to the Windows Script Host (`Windows\System32\wscript.exe`).
- Open and display documents through Windows rather than just returning them directly to the browser. The `TemplateRunner.ScriptControl` control supports this. The default whitelist version of the control directly opens documents that have an allowed file type. The blacklist version of the control opens any file type except for forbidden file types. See “Whitelist or Blacklist Versions” on page 28.

Related ActiveX Objects

The control itself spawns a number of other ActiveX objects. The following table lists the objects and how ClaimCenter uses them:

`ADODB.Stream`	Writes binary contents to files.
`MSXML2.XMLHTTP`	Retrieves document contents directly from the Guidewire server.
`Scripting.FileSystemObject`	Creates and finds files.
`Shell.Application`	Opens files in the appropriate application.

`WScript.Shell`	Supports running JScript files on Windows.

The Windows OS and MDAC version 2.5 or higher provide these objects.

Support for Document Management Systems

ClaimCenter provides a **Documents** section for certain entity types such as a Claim. The **Documents** section lists a set of document references that ClaimCenter 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 `ClaimCenter/java-api/examples/src/examples/plugins/document`. If you do not see the `java-api` directory, run the following command from the ClaimCenter `bin` directory:

```
gwcc regen-java-api
```

You can create a custom integration with a document management system. For details, see “Document Management” on page 249 in the *Integration Guide*.

Enabling Guidewire Document Assistant

The Guidewire Document Assistant is enabled by default. The following procedure enables the Guidewire Document Assistant if it is disabled.

As a user initiates a ClaimCenter session, the browser downloads and installs the `TemplateRunner.ScriptControl` ActiveX control, unless you configure ClaimCenter to disable the control. Depending on the browser settings, ClaimCenter might prompt the user to install the Guidewire Document Assistant. This name is synonymous with the `TemplateRunner.ScriptControl` ActiveX control.

To enable the Guidewire Document Assistant

1. Enable the ActiveX control from the `config.xml` file by setting:
   ```
   <param name="AllowActiveX" value="true"/>
   ```
2. Configure ClaimCenter to use the Guidewire Document Assistant ActiveX control to display documents by setting:
   ```
   <param name="UseGuidewireActiveXControlToDisplayDocuments" value="true"/>
   ```
3. Save `config.xml`.

The first time a user attempts to launch a document or document template from ClaimCenter, the browser downloads the `TemplateRunner.ScriptControl` ActiveX control file `TemplateRunner.ocx`.

Disabling and Removing Guidewire Document Assistant

ClaimCenter enables the Guidewire Document Assistant ActiveX control by default. Your organization might decide not to use ActiveX. Guidewire does not recommend disabling the control because ClaimCenter includes document management features that rely on the control.

The following procedures disable the Guidewire Document Assistant and remove the ActiveX control from user machines.

To disable the Guidewire Document Assistant

1. Disable the ActiveX control from the `config.xml` file by setting:
   ```
   <param name="AllowActiveX" value="false"/>
   ```
2. Configure ClaimCenter to not use the Guidewire Document Assistant ActiveX control to display documents by setting:
   ```
   <param name="UseGuidewireActiveXControlToDisplayDocuments" value="false"/>
   ```

3. Save `config.xml`.

If users have connected to ClaimCenter while the Guidewire Document Assistant ActiveX control was enabled, remove the control from each user machine.

To remove the ActiveX control from a user computer

1. Close any open ClaimCenter sessions on the client computer.
2. Open Internet Explorer.
3. Click **Tools** → **Manage Add-ons**.
4. Under **Add-on Types**, select **Toolbars and Extensions**, if not already selected.
5. Under **Show:** select **All add-ons**.
6. Under **Guidewire Software**, right-click **TemplateRunner.ScriptControl** and select **More Information**.
7. Click **Remove**.
8. Open Windows Explorer.
9. Navigate to the `Windows\Downloaded Program Files` (Windows XP) folder or the `Winnt\Downloaded Program Files` (Windows 2000) folder.
10. Delete the `TemplateRunner` files.

The built-in document-management features of ClaimCenter rely on the ActiveX control. If you do turn off the control, the user interface controls related to these features are still in the interface. Configure PCF files to remove or modify the controls.

For example, the default `NewTemplateDocumentDV.pcf` includes a **Create Document** button. If you disable the ActiveX control, then document creation occurs on the client machine rather than the server. Therefore, set the `download` property of the **Create Document** button to `true`.

To modify the button

1. Open Guidewire Studio by executing the following command from `ClaimCenter/bin`:
   ```
   gwcc studio
   ```
2. Click **Page Configuration (PCF)** → **document** → **NewTemplateDocumentDV**.
3. Click the **Create Document** button element.
4. On the **Properties** tab, under **Advanced properties**, change the **download** property from **false** to **true**. If you have not previously modified `NewTemplateDocumentDV.pcf`, Studio prompts you before copying the file to the configuration module. Click **Yes**.

Repeat this procedure for any custom pages you have added that have a similar **Create Document** button.

Configuration Parameters for the Guidewire Document Assistant

You can set the following configuration parameters related to document management:

Parameter	Description

`AllowActiveX`	Whether to allow ActiveX controls in the ClaimCenter interface (for document management, for example). 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` • `UseGuidewireActiveXControlToDisplayDocuments` **Default:** `True`
`AllowActiveXAutoInstall`	Whether ClaimCenter automatically attempts to install the Guidewire Document Assistant and supporting JScript files. If `AllowActiveXAutoInstall` is set to `false`, ClaimCenter does not attempt to install Guidewire Document Assistant and supporting JScript files. You must manually install the Guidewire Document Assistant and supporting JScript files if you want to use it. ClaimCenter does not attempt to download and install the control if it is not already present. Contact Guidewire Support for assistance with manual installation of Guidewire Document Assistant. **Default:** `True`
`UseGuidewireActiveXControlToDisplayDocuments`	Determines whether the server uses the `TemplateRunner.ScriptControl` control. If `false`, ClaimCenter does not use the control and document contents return directly to the browser.
`DisplayDocumentEditUploadButtons`	Turns on or off the display of edit and upload buttons in the document list. By default, the value is `true` and the buttons appear.
`DocumentContentDispositionMode`	Specifies how the browser displays a document. This value can either be `inline` (the default) or `attachment`.
`DocumentTemplateDescriptorXSDLocation`	Name of the XSD file ClaimCenter uses to validate doc template descriptor XML files. ClaimCenter loads this file first from `ClaimCenter/modules/configuration/config/resources/doctemplates`. If ClaimCenter does not find the XSD file in that location, ClaimCenter searches for the base version in `ClaimCenter/modules/cc/config/resources/doctemplates`.
`MaximumFileUploadSize`	Specifies the maximum file size in megabytes that a user can upload. 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.

Whitelist or Blacklist Versions

Guidewire provides two versions of the Guidewire Document Assistant. The default version uses a whitelist of file types that the `TemplateRunner.ScriptControl` control opens and displays. The default control supports auto-launching of the following file types:

- AVI
- BMP
- CSV
- DOC
- DOCX
- GIF
- HTM
- HTML
- JPG
- LOG
- MDI
- MOV
- MPEG
- MPG
- PDF
- PNG
- PPS
- PPT
- PPTX
- PS
- RTF
- RTX
- TIF
- TIFF
- TXT
- WAV
- WMA
- XLS
- XLSX
- XML

For files with extensions not listed, the default control 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, refer to 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`

The blacklist version of the Guidewire Document Assistant uses a list of known potentially dangerous file types and a configurable whitelist. The default whitelist includes the same file types as the whitelist version of the ActiveX control.

The control does not launch any file of a type that is on the blacklist, regardless of if the file is on the configurable whitelist. The browser still prompts the user to download the file, but the Guidewire Document Assistant does not launch the file. If a file type is on the whitelist and not on the blacklist, Guidewire Document Assistant downloads the file and launches it using the associated program. If a file type is not on either list, then the Guidewire Document Assistant downloads the file but does not launch the file. Use anti-virus and file system protection software to minimize the risk from downloaded files.

WARNING The blacklist version of the control is highly insecure, known to be a CVSS (Common Vulnerability Scoring System) high vulnerability security problem. Only use the blacklist control in trusted environments that also use other protection mechanisms.

File types that the blacklist version does not open include:

File type	Description
BAS	Visual Basic class module
BAT	Batch file
CHM	Compiled HTML help file
CMD	Windows NT command script
COM	MS-DOS application
CPL	Control Panel extension
DLL	Dynamic link library
EXE	Application
HTA	HTML application
INF	Setup information file
INS	Internet communication settings
ISP	Internet communication settings
JAR	Java jar executable
JS	JScript file
JSE	JScript encoded script file
LNK	Shortcut
MSI	Windows installer package
MSP	Windows Installer patch
MST	Visual test source file
OCX	ActiveX objects
PIF	Shortcut to MS-DOS program
PL, PERL	Perl script
REG	Registration entries
SCR	Screen saver
SCT	Windows script component

File type	Description
SHS	Shell scrap object
SYS	System configuration file / driver
URL	Internet shortcut (Uniform Resource Locator)
VB	VBScript file
VBE	VBScript encoded script file
VBS	VBScript script file
WSC	Windows script component
WSF	Windows script file
WSH	Windows scripting host settings file

By default ClaimCenter uses the whitelist version of the control.

To configure ClaimCenter to use the blacklist ActiveX control

1. Launch Guidewire Studio by running the following command from `ClaimCenter\bin`:
   ```
   gwcc studio
   ```
2. In the Studio **Resources** pane, select **configuration** → **Web Resources** → **web/templates** → **document** → **DocumentControl.gs**.
3. Within the `OBJECT` block, cut the `CLASSID` and `CODEBASE` values and paste these to the commented block above the `OBJECT`. The `CODEBASE` to move references `GuidewireDocumentAssistant.CAB`. This preserves the `CLASSID` and `CODEBASE` values for the whitelist version of the control in the commented section, in case you later want to switch controls.
4. Copy the `CLASSID` and `CODEBASE` values that reference `GuidewireDocumentAssistantConfigurable.CAB` from the commented block above the `OBJECT`.
5. Paste the `CLASSID` and `CODEBASE` values that reference `GuidewireDocumentAssistantConfigurable.CAB` into the `OBJECT` definition. Only copy the `CLASSID` and `CODEBASE` lines.
6. Save your changes to `DocumentControl.gs`.

To configure allowed file types for the blacklist ActiveX control

1. Check that the file type that you want to add is not on the blacklist provided previously in this section.
2. On the application server, open the `ClaimCenter\webapps\cc\resources\documentassistant\WhitelistExtensions` file. This file contains a dot-delimited list of file extensions.
3. Add the file extension that you want to allow to the end of the list. Do not add an extension for a file type that is already on the blacklist.
4. Save `WhitelistExtensions`.
5. Add a `mimetype` entry to `config.xml` for the new file type.
   ```
   <mimetype name="name" extensions="extension|extension2" icon="icon image" description="description"/>
   ```
   See the `mimetype` definitions in `config.xml` for examples and “Configuring Document Production and MIME Types” on page 256 in the *Integration Guide*.
6. Launch Guidewire Studio by running the following command from `ClaimCenter\bin`:
   ```
   gwcc studio
   ```
7. In the Studio **Resources** pane, select **configuration** → **plugins** → **document** → **IDocumentProduction**.
8. Under **Parameters**, click **Add**. If you have not yet customized `IDocumentProduction`, Studio prompts you to create a copy of `IDocumentProduction` in the configuration module.

9. Set the **name** of the new parameter to the name you provided for the `mimetype` entry in `config.xml`.
10. Set the **value** of the new parameter to `com.guidewire.cc.plugin.document.internal.GosuDocumentProductionImpl`.
11. Save `IDocumentProduction`.
12. On the development workstation, delete the contents of `C:\Users\`*`user`*`\AppData\Local\Temp\Guidewire Scripts`.
13. Restart the server by running the following command from `ClaimCenter\bin`:
    ```
    gwcc dev-start
    ```
14. Log into ClaimCenter.
15. Open `C:\Users\`*`user`*`\AppData\Local\Temp\Guidewire Scripts\`*`version`*`\WhitelistExtensions` in a text editor.
16. Verify that the file extension that you added appears in the `WhitelistExtensions` file.
17. To deploy to a production environment, repackage and redeploy the WAR or EAR file. See "Deploying ClaimCenter to the Application Server" on page 56 in the *Installation Guide*.
18. Delete the old ActiveX control from all client machines.
    a. Close any open ClaimCenter sessions on the client computer.
    b. Open Internet Explorer.
    c. Click **Tools** → **Manage Add-ons**.
    d. Under **Add-on Types**, select **Toolbars and Extensions**, if not already selected.
    e. Under **Show:** select **All add-ons**.
    f. Under **Guidewire Software**, right-click **TemplateRunner.ScriptControl** and select **More Information**.
    g. Click **Remove**.
    h. Open Windows Explorer.
    i. Navigate to the `Windows\Downloaded Program Files` (Windows XP) folder or the `Winnt\Downloaded Program Files` (Windows 2000) folder.
    j. Delete the `TemplateRunner` files. The browser reinstalls the control the next time the user opens the browser.

Specifying Location for Guidewire Document Assistant Scripts

The Guidewire Document Assistant control checks for a key in the Windows registry: `HKEY_LOCAL_MACHINE\SOFTWARE\Guidewire\Document Assistant` and a value within that key named `Script Directory`. The `Script Directory` value specifies where the control searches for scripts and the `WhitelistExtensions` file. If `AllowActiveXAutoInstall` is `true`, the `Script Directory` value also specifies the location where the control installs scripts and the `WhitelistExtensions` file. The value of the key can be of type `REG_SZ` or `REG_EXPAND_SZ`. Use the `REG_EXPAND_SZ` type if the value contains references to environment variables.

If `Script Directory` is either not present or is set to the empty string and `AllowActiveXAutoInstall` is `true`, both the control and supporting scripts and `WhitelistExtensions` can be downloaded. ClaimCenter installs scripts and `WhitelistExtensions` in a path within `%TEMP%`, specifically `%TEMP%\Guidewire Scripts\`*`version`*. The *`version`* is a number that changes with each ClaimCenter version.

If `Script Directory` is set to a non-empty string, the control does not try to download its supporting scripts, regardless of the `AllowActiveXAutoInstall` setting. With `AllowActiveXAutoInstall` set to `true` and the

registry value set, end users are allowed to download and automatically install the control itself. However, the scripts are manually installed in a fixed location and never downloaded. The control searches for scripts and the whitelist file in a path specified by `Script Directory`. For example, if you set `Script Directory` to `C:\Guidewire\Document Assistant`, the control searches for scripts in `C:\Guidewire\Document Assistant\Guidewire Scripts\`*`version`*.

The following table describes use cases. The first and last use cases in the following table are the most common scenarios.

Value of `AllowActiveXAutoInstall`	**Value of** `Script Directory`	**Behavior**
`true`	Not set or set to empty string	Both the Guidewire Document Assistant TemplateRunner control and supporting scripts (and `WhitelistExtensions`) can be downloaded. ClaimCenter automatically installs scripts (and `WhitelistExtensions`) in `%TEMP%\Guidewire Scripts\`*`version`*.
`true`	Set to non-empty string	ClaimCenter can download the control but cannot download supporting scripts (and `WhitelistExtensions`). Scripts (and `WhitelistExtensions`) must be manually installed in the following location: *`Script Directory`*`\Guidewire Scripts\`*`version`*.
`false`	Not set or set to empty string	Neither the control nor supporting scripts (and `WhitelistExtensions`) can be downloaded. They must be manually installed. Scripts (and `WhitelistExtensions`) must be manually installed in `%TEMP%\Guidewire Scripts\`*`version`*. While this configuration is possible, it is unlikely to be used.
`false`	Set to non-empty string	Neither the control nor supporting scripts (and `WhitelistExtensions`) can be downloaded. They must be manually installed. Scripts (and `WhitelistExtensions`) must be manually installed in the following location: *`Script Directory`*`\Guidewire Scripts\`*`version`*.

Troubleshooting Guidewire Document Assistant Problems

This topic describes procedures to troubleshoot issues with the Guidewire Document Assistant.

Updating Whitelist Extensions File

Whenever the whitelist or blacklist changes, ClaimCenter must download a new version of the `WhitelistExtensions` file on user computers. To force the download to happen, remove the local `WhitelistExtensions` file and attempt to launch a document template from ClaimCenter. The browser reinstalls the `WhitelistExtensions` file. The following procedure describes updating the `WhitelistExtensions` file on a user's computer.

To update the whitelist extensions file

1. Close any open ClaimCenter sessions on the client computer.
2. Open Windows Explorer.
3. Navigate to the `C:\Users\`*`user`*`\AppData\Local\Temp\Guidewire Scripts\`*`version`* directory.
4. Delete the `WhitelistExtensions` file.
5. Refresh the ClaimCenter application in the browser. ClaimCenter downloads the latest `WhitelistExtensions` file.

Adding ClaimCenter Server to Trusted Sites

Windows Vista and Windows 7 users must add the ClaimCenter server to the list of trusted sites or they might not be able to create documents from templates.

To add the ClaimCenter server to trusted sites

1. Open Internet Explorer.
2. Select **Tools** → **Internet Options**.
3. Click the **Security** tab.
4. Click the **Trusted sites** icon.
5. For Internet Explorer 7, ensure that **Enable Protected Mode** is not checked. By default, this setting is unchecked for trusted sites.
6. Click **Sites**.
7. Uncheck **Require server verification (https:) for all sites in this zone**. Alternatively, run ClaimCenter using `https`.
8. Enter the ClaimCenter server name.
9. Click Add.
10. Close all open Internet Explorer windows.
11. Restart Internet Explorer.

Consider creating a REG file, script, SMS package, or similar to make this change on a number of client systems.

Configuring an Email Server for Notifications

ClaimCenter 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 or notes a claim exception.

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. ClaimCenter 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 ClaimCenter for email notifications, configure the plugin parameters in Guidewire Studio.

To configure email plugin parameters in Guidewire Studio

1. Open Guidewire Studio by running `gwcc studio` from the `ClaimCenter/bin` directory.
2. In the **Resources** pane, select **configuration** → **Plugins** → **gw** → **plugin** → **messaging** → **MessageTransport** → **emailMessageTransport**.
3. If the **Enabled** checkbox is not selected, select the checkbox to enable the plugin.
4. Edit the values set to the following parameters:
   - `smtpPort`
   - `smtpHost`
   - `defaultSenderName`
   - `defaultSenderAddress`
5. Click **File** → **Save Changes**.
6. Restart the ClaimCenter server.

The *ClaimCenter Integration Guide* discusses using templates to generate email messages in more detail.

Changing the Unrestricted User

By default, ClaimCenter 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 3

Configuring Logging

This topic discusses logging in ClaimCenter.

This topic includes:

Logging Overview

The `ClaimCenter/modules/configuration/config/logging/logging.properties` file specifies system logging options for the ClaimCenter application server. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources** → **logging**.

ClaimCenter uses version 1.2.15 of the Java `log4j` logging facility. The `logging.properties` file uses the format specified by `log4j`. The entries in `logging.properties` control what to log and in which file to write the log. The setting that controls basic logging looks like the following:

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

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

Within `logging.properties`, entries like `log4j.appender.*` indicate the parameters of each output point. These entries identify properties such as location or output format options. As ClaimCenter starts, it attempts to write a log file in the location specified by `log4j.appender.DailyFileLog.File`.

ClaimCenter creates the log file automatically. By default, ClaimCenter writes the log to the directory specified by the property `DailyFileLog.File`. By default, this directory is `tmp/gwlogs/ClaimCenter/logs/cclog.log`. However, if the directory specified by `DailyFileLog.File` does not exist, ClaimCenter writes log information only to the console.

Notes

- See `http://logging.apache.org/log4j/1.2/index.html` for detailed information about creating log4j entries.
- If the `env` environment property is non-null, ClaimCenter tries to obtain the logging configuration from an *`env`*`-logging.properties` file in `ClaimCenter/modules/configuration/config/logging`. For example, if you have an environment called test, ClaimCenter looks for logging properties in `test-logging.properties`. If this file does not exist or if `env` is `null`, then the logging configuration is taken from the default `logging.properties` file. See "Defining the Application Server Environment" on page 18 for information on the `env` property.
- After you edit the `logging.properties` file, you must rebuild and redeploy the ClaimCenter application file for the changes to take effect.

Logging for Studio

You can also enable diagnostic logging for Guidewire Studio. You configure logging for Studio in a different file, `studio.properties`, rather than `logging.properties`. Both files use the format specified by `log4j`. Guidewire disables diagnostic logging for Studio by default. Typically you do not need to enable Studio logging. However, you might need to enable logging for Studio at the request of Guidewire Support. See "Configuring Diagnostic Logging in Studio" on page 90 in the *Configuration Guide*.

Specifying Location of Log Files for the View Logs Page

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

See "View Logs" on page 159.

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 using variables

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

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

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

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

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

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

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

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

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

   See “Using the registry Element to Specify Environment Properties” on page 18.

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

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

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

Key Logging Options

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

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

Level	Description
`TRACE`	Provides messages about processes that are about to start or that completed. Trace logging has no or minimal impact on system performance.
`DEBUG`	Messages that test a provable and specific theory intended to reveal some system malfunction. For example dumping the contents of an XML document is fine.
`INFO`	Messages intended to convey a sense of correct system operation such as a component started, a user logged on or off ClaimCenter, and so forth.
`WARN`	Messages that denote something potentially out of ordinary is happening. Examples include: assignment rules that do not end in an assignment, a plugin call that took 90 seconds, and so forth.
`ERROR`	Messages that denote something is wrong. For example, a remote system has refused a connection within a plugin, or ClaimCenter can not complete some operation even with a default.

Set the logging level property of the different logging entries to set how much information you want sent to each type of log. Setting the value on the `RootLogger` causes any children to inherit the same level, provided those children have not set a different logging level. Refer to log4j documentation for a complete discussion of logging levels and inheritance.

Logging Successfully Archived Claims

ClaimCenter creates a separate log for successfully archived objects. Each log is unique to a run of the archive work queue. The log contains a list of the claims that were successfully archived.

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

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

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

# Logger for successful archived claims
# log4j.category.Server.Archiving.Success=INFO, ArchivedClaimsLog

# Logger for debugging the Claim graphs of claims being archived
# log4j.category.Server.Archiving.Graph=DEBUG, ArchivedClaimsLog

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

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

Logs for Additional System Components

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

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

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

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

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

Configuring Information in Log Messages

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

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

Character	Description
%%	Writes the percent sign to output.
%c	Name of the logger category. See “Listing Logger Categories” on page 40 for categories provided with ClaimCenter.
%C	Name of the Java class. Because the ClaimCenter logging API is a wrapper around log4j, `%C` returns `com.guidewire.logging.Logger`. If you want class names in your log messages, include them specifically in the message rather than using `%C` in the conversion pattern.

Character	Description
%d	Date and time. Acceptable formats include: • %d{ISO8601} • %d{DATE} • %d{ABSOLUTE} • %d{HH:mm:ss,SSS} • %d{dd MMM yyyy HH:mm:ss,SSS} • and so on. ClaimCenter uses %d{ISO8601} by default.
%F	Name of the Java source file. Because the ClaimCenter logging API is a wrapper around log4j, %F returns a file name for the ClaimCenter logging API. If you want file names in your log messages, include them specifically in the message rather than 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 ClaimCenter logging API is a wrapper around log4j, the information returned is for the ClaimCenter logging API. If you want information such as class and method names in your log messages, include them specifically in the message rather than using %l in the conversion pattern.
%L	Line number in Java source. Because the ClaimCenter logging API is a wrapper around log4j, %L returns a line number from the ClaimCenter logging API. If you want line numbers in your log messages, include them specifically in the message rather than using %L in the conversion pattern.
%m	The log message.
%M	Name of the Java method. Because the ClaimCenter 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 using %M in the conversion pattern.
%n	Newline character of the operating system. This is preferable to entering \n or \r\n as it works across platforms.
%p	Priority of the message. Typically, either FATAL, ERROR, WARN, INFO or DEBUG. You can also create custom priorities in your own code.
%r	Number of milliseconds since the program started running.
%t	Name of the current thread.
%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 For example, to include the server name, add %X{server}. The user key lists a sequence number assigned to the user by the server and is not very informative. To include user login ID information, instead use the userID key.

Formatting Log Messages

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

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

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

Listing Logger Categories

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

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

Or, you can use the SOAP API ISystemToolsAPI.getLoggingCategories.

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

The system_tools command and SOAP API only list logging categories defined by ClaimCenter. Some third-party components, such as JGroups, provide their own categories.

For information about logging for plugins, see “Logging” on page 293 in the *Integration Guide*.

The following logging categories exist for all Guidewire applications:

Category	Description
AddressAutoSync	Logging for the synchronization of contact data between ContactCenter and ClaimCenter.
AddressCorrection	Logging for the address correction performed by the Geodata implementation.
Api	Used for all SOAP API calls.
Application	Internal Guidewire base logger category for application logging.
Application.Addressbook	Used for Address Book subsystem. NOTE: This is not the category used for ContactCenter. Rather, Guidewire intends this category for use by platform code that interacts with the Address Book.
Application.Addressbook.Config	Logging for the configuration of the Address Book subsystem.
Assignment	Internal Guidewire base logger category for assignment logging.
Availability	Logging of the determination of availability of elements in ClaimCenter. Availability criteria for each element are based on the element type. For example, one criterion could be the effective date of an element. If the effective date is not prior to or equal to today’s date, the element would not be available.
Configuration	Logging for configuration problems in such areas as in security configuration, PCF configuration, locale configuration and so forth.
Configuration.geodata	Logging for configuration issues related to the Geodata daemon.
Datagen	Internal Guidewire logger category for testing.
GeneralExecution	The default logger category for logging done by the gw.api.util.Logger class.
Geodata	Base logging category for the Geodata daemon.
Import	Logging for import of XML data into ClaimCenter.
Messaging	Base logging category for the messaging system.

Category	Description
Messaging.Email	Logging of email message destination and email generation code.
Messaging.Events	Logging of events. For more information about events and messaging, see “Messaging and Events” on page 139 in the *Integration Guide*.
Plugin	Logging for all calls into any plugin. This is also the “parent category” for individual plugin categories. For more information on logging with plugins, see “Logging” on page 293 in the *Integration Guide*.
Plugin.IAddressBookAdapter	Logging for the address book plugin, which ClaimCenter uses to communicate with ContactCenter. For more information on `IAddressBookAdapter`, see “Address Book and Contact Search Plugins” on page 301 in the *Integration Guide*.
Plugin.IApprovalAdapter	Logging for the approval plugin, which ClaimCenter uses to determine whether items need approval and to whom ClaimCenter assigns the item if it does need approval. For more information on `IApprovalAdapter`, see “Approval Plugin” on page 335 in the *Integration Guide*.
Plugin.IContactSearchAdapter	Logging for the address book plugin, which ClaimCenter uses for search and retrieval of contact data. For more information on `IContactSearchAdapter`, see “Address Book and Contact Search Plugins” on page 301 in the *Integration Guide*.
Plugin.ICustomPickerAdapter	Logging for the custom picker plugin. This plugin is deprecated and will be removed in a future release.
Plugin.IDocumentMetadataSourceBase	Logging for the documentation management metadata plugin. For more information on documentation management plugins, see “Document Management” on page 249 in the *Integration Guide*.
Profiler	Used to log the Guidewire built-in profiler.
RuleEngine	Do not use. Use `RuleExecution` instead.
RuleExecution	Logging done in rules.
Rules	Do not use. Use `RuleExecution` instead.
Security	Internal Guidewire base logger category for security logging.
Server	Internal Guidewire base logger category for server/platform logging.
Server.Admin Export/Import	Logging for the process of importing and exporting of administrative data.
Server.Archiving	Logging for messages generated during both graph construction and validation and by workers in the archive work queue.
Server.Archiving.Success	Logging for messages generated for each entity that is successfully archived.
Server.BatchProcess	Batch process logging. Starting, stopping, and so forth. You can not configure a particular batch process to use a different logger.
Server.Cluster	Logging for ClaimCenter messages related to clustering. The clustering implementation, JGroups, has its own logging category, `org.jgroups`. The default `logging.properties` file includes a section for this category.
Server.ConcurrentDataChangeException	Logging for concurrent data changes, in which two commits to the database alter the same object.
Server.Database	Used for all database type logging, transactions and so forth.
Server.Database.Staging	Used for staging table related logging.
Server.Global Cache	Logs messages related to what is happening in the cache. ClaimCenter only uses this category to print warnings and errors.
Server.Preload	Provides DEBUG level logging of all actions during server preloading of Gosu classes. See “Preloading Gosu Classes” on page 109 in the *Configuration Guide*.
Server.Profiler	Logging for the Guidewire built in Profiler.

Category	Description
Server.RunLevel	Used by “startables” during their start and stop methods and to log run level changes.
Server.Workflow	The base logger category for all workflow engine logging.
Server.Workflow.WorkflowResourceController	Logging of errors on failure to parse a workflow definition file.
Studio	Internal Guidewire base logger category for Guidewire Studio logging.
Test	Guidewire internal category.
UserInterface	Internal Guidewire base logger category for user interface logging.
UserInterface.Performance	Logging for events such as frame rendering and widget events. To log user interface performance messages, set the logging level for the UserInterface.Performance category to TRACE. All events contain a frame reference. Messages regarding frame rendering are tagged with the text PERFLOG-RENDER. Messages about widget events are tagged with the text PERFLOG-WIDGET. The timestamp and user ID are available from the context. You can configure the UserInterface.Performance logging category to list the user ID and timestamp by including the fields %X{userID} and %d in the layout conversion pattern in logging.properties. PCF element information is available within the widget event messages.
Workqueue	Base logging category for work queue functionality. For more information on work queues, see “Batch Processes and Work Queues” on page 129.
Workqueue.Instrumented	Trace logging of commits to the instrumented worker table.
Workqueue.Item	Logging for processing of work queue work items.
Workqueue.Runner	Logging of informational messages for a worker, including time it starts, stops and suspends, various errors while running, and a trace level for each execution.

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

Name	Description
Application.ClaimUsers	Logging for the claim users functionality, especially the related contacts feature.
Application.Financials	Logging for the financials subsystem.
Application.NewClaimWizard	Logging for the new claim wizard.
Application.Segmentation	Logging for the segmentation engine, used to segment new claims and exposures. Segmentation is done either by rules or by the segmentation plugin.
Application.Snapshot	Logging for creating or displaying a claim snapshot.
Messaging.Metro	Logging of messages to and from Metropolitan Reporting Bureau.See “Metropolitan Reports” on page 353 in the *Application Guide*. For information about integrating with Metropolitan Reporting Bureau, see “Metropolitan Reporting Bureau Integration” on page 389 in the *Integration Guide* For more information about Metropolitan Reporting Bureau services, refer to their web site: http://www.metroreporting.com
Plugin.IAssignmentAdapter	Logging for assignment plugin, alternative to assigning by rules.
Plugin.IClaimNumGenAdapter	Logging for the claim number generator, used to generate claim numbers for new claims.
Plugin.ICustomConditionAdapter	Logging for the deprecated custom condition plugin, now replaced by Gosu plugins.

Name	Description
Plugin.IDeductionAdapter	Logging for the policy search plugin, used to create deductions from a new payment.
Plugin.IInitialReserveAdapter	Logging for the initial reserve plugin, used to set the initial reserve for a new exposure.
Plugin.IPolicyRenewalAlertAdapter	Logging for the policy renewal plugin, used to create new policy renewal alerts.
Plugin.IPolicySearchAdapter	Logging for the policy search plugin, used to search for and to retrieve policies from the policy system.
Plugin.ISegmentationAdapter	Logging for the segmentation plugin, used to segment new claims and exposures.

Making Dynamic Logging Changes without Redeploying

You can make an immediate logging change in the ClaimCenter server without having to first redeploy ClaimCenter. Use the `admin/bin/system_tools` command-line utility or ClaimCenter API. Use the following command:

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

You must provide the name of the logger to update. Specify each logger name in a `log4j.category.string` entry in the `logging.properties` file. Use the *string* to identify the logger. For example, the name of the logger for the Approval Engine is `com.guidewire.cc.server.approval`. To update the logging level for the `DailyFileLog` or root logger category, use the name `RootLogger`.

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

chapter 4

Configuring and Maintaining the ClaimCenter Database

This topic discusses key issues for configuring and maintaining the ClaimCenter database.

This topic includes:

See also

- "Configuring the Database" on page 20 in the *Installation Guide*
- "Configuring a Database Connection" on page 44 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:

- 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 ClaimCenter. This can cause the data distribution tool to fail and cause other problems. See “Guidewire Database Direct Update Policy” on page 46.
- Do not add lots of `mediumtext` and `CLOB` columns to a table.
- Do not add an index outside of ClaimCenter 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 ClaimCenter does not run out of space.
- Back up the database periodically to support disaster recovery options. See “Backing up the ClaimCenter Database” on page 59.
- Update database statistics periodically so that the query optimizer selects an efficient plan for executing application queries. See “Configuring Database Statistics” on page 53.
- Run consistency checks on the database, especially after importing data. See “Checking Database Consistency” on page 51.

Guidewire Database Direct Update Policy

ClaimCenter 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 ClaimCenter. The internal application logic, embedded in ClaimCenter 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 ClaimCenter 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 ClaimCenter 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 ClaimCenter 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, ClaimCenter 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 parameters to the `<database>` element in the `config.xml` file. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**.

Parameter	Description
`maxActive`	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 ClaimCenter at the same time. If this is set to a negative number, then there is no limit. The default value is `-1`.
`maxIdle`	The maximum number of idle connections that the application server maintains. If this is set to a negative number, then there is no limit. The default value is `-1`.
`maxWait`	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`.
`testWhileIdle`	Whether ClaimCenter performs eviction runs on the connection pool. During an eviction run, ClaimCenter scans the connection pool and tests a number of idle connections equal to `numTestsPerEvictionRun`. If a connection has been idle more than `minEvictableIdleTimeMillis` milliseconds, ClaimCenter evicts the connection from the pool. Otherwise, ClaimCenter executes a simple query on the connection. If the query fails with an exception, then ClaimCenter evicts the connection. You can configure the validation query by adding a `validationQuery` parameter. The default value of `testWhileIdle` is `true`.
`numTestsPerEvictionRun`	The number of idle connections ClaimCenter tests each eviction run. The default is `3`.
`timeBetweenEvictionRunsMillis`	The time, in milliseconds, that ClaimCenter waits between eviction runs. The default is `60000`.
`minEvictableIdleTimeMillis`	The time, in milliseconds, that a connection can be idle before ClaimCenter considers the connection idle and eligible for eviction. The default is `300000`.
`testOnBorrow`	Whether ClaimCenter tests a connection by running a simple query as ClaimCenter first borrows the connection from the connection pool. The default is `false`.
`testOnReturn`	Whether ClaimCenter tests a connection by running a simple query as ClaimCenter returns the connection to the connection pool. The default is `false`. Since ClaimCenter 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.
`validationQuery`	The validation query ClaimCenter runs to test a connection. The default is "`select 1 where 1 = 0`" on SQL Server, and "`select 1 from dual where 1 = 0`" on Oracle.
`stmtPool.enabled`	This must always be `false` for both Oracle and SQL databases. For non-Guidewire uses, it enables a JDBC implementation to cache and reuse prepared statements.

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

The parameters listed previously apply if 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 ClaimCenter to Use a JNDI Data Source" on page 48 in the *Installation Guide* for information on configuring ClaimCenter to use the application server connection pool.

Setting Search Parameters for Oracle

The execution plan that Oracle uses for certain searches performs poorly. To remedy this issue, Guidewire includes configuration parameters in the `config.xml` file. These Boolean parameters modify the Oracle configuration when performing claim or team group activity searches. By default, all of these parameters are set to `true`. None of these parameters has an effect on database types other than Oracle. The configuration parameters are described in the following table.

Parameter	Description
`DisableIndexFastFullScanForClaimSearch`	ClaimCenter works around Oracle bug 5886252 by disabling index fast full scan when executing certain claim searches on Oracle. If a future version of Oracle fixes the defect this parameter may be removed.
`DisableCBQTForClaimSearch`	ClaimCenter works around Oracle bug 6990305 by disabling optimizer cost base transformation when executing certain claim searches on Oracle. If a future version of Oracle fixes the defect this parameter may be removed.
`DisableHashJoinForClaimSearch`	ClaimCenter works around hash join related query plan problems when executing certain claim searches on Oracle.
`DisableSortMergeJoinForClaimSearch`	ClaimCenter works around sort merge join query plan problems when executing certain claim searches on Oracle.
`SetSemiJoinNestedLoopsForClaimSearch`	ClaimCenter works around semi join query plan problems by forcing nested loop semi join instead of choose when executing certain claim searches on Oracle.
`DisableIndexFastFullScanForTeamGroupActivities`	ClaimCenter works around Oracle bug 5886252 by disabling index fast full scan when executing certain team group activity searches on Oracle. If a future version of Oracle fixes the defect this parameter may be removed.
`DisableCBQTForTeamGroupActivities`	ClaimCenter works around Oracle bug 6990305 by disabling optimizer cost base transformation when executing certain team group activity searches on Oracle. If a future version of Oracle fixes the defect this parameter may be removed.
`DisableHashJoinForTeamGroupActivities`	ClaimCenter works around hash join related query plan problems when executing certain team group activity searches on Oracle.
`DisableSortMergeJoinForTeamGroupActivities`	ClaimCenter works around sort merge join query plan problems when executing certain team group activity searches on Oracle.

Using Oracle Materialized Views for Claim Searches

It is possible to create materialized views in an Oracle schema to improve the performance of queries that ClaimCenter runs as part of a Claim search operation. Materialized views can be useful if performing a search for a claimant or for any involved party using the name of a person or a company. If you implement materialized

views in the ClaimCenter schema, then Oracle attempts to use these materialized views if a re-written query block matches the text defined in the view.

Guidewire provides the configuration parameter `QueryRewriteForClaimSearch` to enable various options for an Oracle query rewrite using materialized view. By setting this parameter, you can force a query to be rewritten using a materialized view or to let the Oracle optimizer make the choice based on the cost calculation.

The following list describes the valid values for this parameter:

Value	Meaning
`FORCE/STALE`	Oracle attempts to rewrite the query using an appropriate materialized view even if the optimizer cost estimate is high. Oracle allows the rewrite even if the data in the materialized is not the same as in the base tables.
`FORCE/NOSTALE`	Oracle attempts to rewrite the query using an appropriate materialized view even if the optimizer cost estimate is high. Oracle ignores the materialized view if the data in the view is not fresh.
`COST/STALE`	If the Oracle cost-based optimizer evaluates the rewrite to be cheaper than other plans, it uses the materialized view. If it is costlier to execute the rewritten path, then Oracle performs a join of the base tables. The rewrite can happen even if the data in the view is stale.
`COST/NOSTALE`	If the Oracle cost-based optimizer evaluates the rewrite to be cheaper than other plans, it uses the materialized view. If it is costlier to execute the rewritten path, then Oracle performs a join of the base tables. If the data in the view is not fresh, Oracle ignores the view and performs the join on the base tables.

Note: If you provide an invalid value, the server ignores it.

Default: None

Disabling Query Rewrites

If you implement materialized views in the ClaimCenter schema, then Oracle attempts to use these materialized views if a rewritten query block matches the text defined in the view. However, the use of materialized views in database queries is not always desirable due to performance considerations.

Thus, ClaimCenter provides an option to disable the rewriting of queries using materialized views. You can disable the use of materialized views in Oracle database queries by setting parameter `queryRewriteEnabled` to `false` in the `<database>` element in `config.xml`. For example:

```
<param name="queryRewriteEnabled" value="false"/>
```

The only valid value for the parameter is `false`:

- If you set this parameter to `false` in `config.xml`, ClaimCenter runs the following statement for each query session:

  ```
  ALTER SESSION SET QUERY_REWRITE_ENABLED = FALSE;
  ```

- If you do not set this parameter in `config.xml` (meaning that you remove the parameter entirely from `config.xml`), then Oracle attempts to use any available materialized view in database queries.

Materialized Views

For a description of how to create materialized views in a ClaimCenter schema, consult the following Guidewire white paper, section 19 (Materialized Views):

ClaimCenter 6.0 using Oracle Database

You can find this document at the following location on the Guidewire Resource Portal:

`https://guidewire.custhelp.com/app/resources/infrastructure/documents`

The **Server Tools** → **Info Pages** → **Oracle AWR Information** page is aware of materialized views. You can use the information on this page to troubleshoot performance problems with the view. However, if the view is refreshed on demand, then the **Oracle AWR Information** page does not capture the refresh queries.

Stale Data

In performance testing, Guidewire observed significant performance degradation if the materialized view was configured to refresh on commit. This is due to a synchronization enqueue required by the refresh process. However, any refresh of the data done outside of the commit operation can potentially display stale data during the search.

Oracle uses a cost-based optimizer approach to determine whether to use a materialized view for a given query. It also expects the data to be fresh for the rewrite. As the refresh process is based on the number of changes to contact and claim contacts, Guidewire strongly recommends that you schedule the refresh process accordingly.

Understanding and Authorizing Database Upgrades

When you start ClaimCenter, 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, ClaimCenter attempts to upgrade the database to match the metadata.

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

Disable automatic upgrades in production environments to prevent unexpected changes. To disable automatic upgrades, 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 ClaimCenter, it must perform a database upgrade.

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

What Happens During a Database Upgrade?

The upgrade 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 ClaimCenter upgrades the database to match the metadata. As the last step in the upgrade, ClaimCenter upgrades the `SystemParameter` entity with the current checksums.

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

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

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

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

The database upgrade takes a number of actions that can impact database statistics. The upgrade might recreate tables, drop indexes and create new indexes. The `autoupgrade` 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 upgrade and the settings of the `database` element's statistic parameters. Run this script after an automatic upgrade to update database statistics. See "Configuring Database Statistics" on page 53 for more information about setting the statistics parameters.

Note: If the server is interrupted during a database upgrade for any reason, the server resumes the upgrade upon restart. The ClaimCenter accomplishes this by storing the steps in the database and marking them completed as part of the same database transaction that applies a change.

For more details on configuration changes that require database upgrades, see the *ClaimCenter Configuration Guide*.

Viewing Detailed Database Upgrade Information

ClaimCenter 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
- 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

Click **Download** to download a ZIP file containing the detailed upgrade information.

To access the Upgrade Info page

1. Start the ClaimCenter 6.0.8 server if it is not already running.
2. Log in to ClaimCenter 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.

Checking Database Consistency

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

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

Running Consistency Checks with System Tools

To launch database consistency checks from the command line, use one of the following commands:

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

The `system_tools` utility is located in the `ClaimCenter/admin/bin` folder. See “system_tools Command” on page 173 for more information.

The `-checkdbconsistency` option runs synchronously, waiting for completion before returning to the command line. The `-checkdbconsistencyasbatch` option runs consistency checks as an asynchronous batch process. Other than that distinction, the options and their optional arguments are the same.

This tool has two optional arguments:

```
-checkdbconsistency tableSelection checkTypeSelection
```

The *tableSelection* argument can be specified as:

- `all` – Run consistency checks on all tables.
- *table name* – The name of a single table on which to run checks.
- `tg.`*table group name* – The name of a table group. Table groups are defined in the `database` element of `config.xml`. For more information, see "Defining Table Groups" on page 45 in the *Installation Guide*.
- `@`*file name* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with `tg.`, such as `tg.MyTableGroup`. You can combine table groups and individual table names in the same file.

The *checkTypeSelection* can be specified as:

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

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

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 *ClaimCenter Data Dictionary* also lists available consistency check types. The typelist is `ConsistencyCheckType`.

Running consistency checks can take a long time. If the connection times out while running this command, run consistency checks on fewer tables at a time by using the arguments shown previously. You can also configure the number of threads to use for consistency checks by modifying `checker.threads` in the `database` block in `config.xml`.

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

You can use the `system_tools -checkdbconsistency` command while there are other users logged into the server, but Guidewire recommends that you first set the ClaimCenter server run level to `MAINTENANCE`. Concurrent users see a performance degradation while the tool is running. See "Using the Maintenance Run Level" on page 64 for more information on server run levels.

Running Consistency Checks from ClaimCenter

You can view and run consistency checks from the **Consistency Checks** page in ClaimCenter. 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 163 for more information.

Running Consistency Checks when the Server Starts

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

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

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.

ClaimCenter provides database statistics generation designed specifically for how the ClaimCenter 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 ClaimCenter to select a bad plan for execution of SQL queries. Therefore, always use ClaimCenter to generate database statistics, rather than by using the statistics generation provided with Oracle. For SQL Server, Guidewire requires that you set `Auto Create Statistics` and `Auto Update Statistics` to `true`.

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 ClaimCenter server often. If you encounter performance problems or degradation related to the database, check the **Database Catalog 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 bi-weekly schedule for updating statistics.

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.

ClaimCenter 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, ClaimCenter 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 ClaimCenter batch processes use work or scratch tables to store intermediate calculations. Other batch processes populate denormalized tables that ClaimCenter 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, ClaimCenter validates the staging tables against the data model and the production tables. To avoid the formation of bad queries prior to loading new data, ClaimCenter 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 176.

Using the statistics page, you can see if any statistics are out of date. See “Database Statistics” on page 165. Guidewire recommends that you archive database statistics as standard practice. This ensures that you have a record of the database history that can be reviewed if necessary.

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

Commands for Updating Database Statistics

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

Type	Description	Command

Full	Generates database statistics for every table in the ClaimCenter database.	To update statistics for all tables:
		`maintenance_tools -password` *`password`* `-startprocess dbstatistics`
		To generate database statistic SQL statements for all tables:
		`maintenance_tools -password` *`password`* `-getdbstatisticsstatements`
		You can use the results purely as a reference, or you can edit the statements and execute them outside of ClaimCenter. 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
		`maintenance_tools -password` *`password`* `-startprocess incrementaldbstats`
		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:
		`maintenance_tools -password` *`password`* `-getincrementaldbstatisticsstatements`
		You can use the results purely as a reference, or you can edit the statements and execute them outside of ClaimCenter. Statements are grouped by table.

Staging	Generates database statistics for staging tables.	`table_import -password` *`password`* `-updatedatabasestatistics`

Configuring Database Statistic Generation

You can control which database statistics statements ClaimCenter generates by configuring the database connection in the `config.xml` file. In the following example, an Oracle database connection shows the use of these parameters:

```
<database name="ClaimCenterDatabase" driver="dbcp" dbtype="oracle" autoupgrade="false"
  automaticallycreatetablespaces="false">
  <param name="jdbcURL" value="jdbc:oracle:thin:USER/PASSWORD@HOSTNAME:PORT:ORACLE_SID"/>
  <databasestatistics samplingpercentage="20" databasedegree="2" >
    <tablestatistics name="cc_check" 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="cc_contact" samplingpercentage="0" databasedegree="4" />
    <tablestatistics name="cc_message" action="delete" />
  </databasestatistics>
  <param name="checker.threads" value="1"/>
  <tablespacemapping logicalname="ADMIN" physicalname="CC_ADMIN"/>
  <tablespacemapping logicalname="OP" physicalname="CC_OP"/>
  <tablespacemapping logicalname="TYPELIST" physicalname="CC_TYPELIST"/>
  <tablespacemapping logicalname="INDEX" physicalname="CC_INDEX"/>
  <tablespacemapping logicalname="STAGING" physicalname="CC_STAGING"/>
</database>
```

The above example configures the following database statistic generation behavior:

- Collect statistics on all ClaimCenter tables using 20% sampling and with a degree of parallelism of 2.
- The dbms_stats command for table cc_check samples 100% and uses a parallel degree of 4.
- The configuration defines a histogram with 254 buckets on cc_check.scheduledsenddate.
- The index on cc_check(publicID, retired) is sampled at 50% and statistics for the index are gathered.
- The dbms_stats command for cc_contact and its indexes will automatically sample.
- ClaimCenter deletes statistics on the cc_message table 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..

You can specify one databasestatistics subelement within a database element. The databasestatistics element has the following format:

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

The following sections discuss this element and how to use it.

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. ClaimCenter 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. The default value is 100. 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. ClaimCenter 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.

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 using the tablestatistics subelement. Setting these 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 ClaimCenter. 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, ClaimCenter on Oracle does not generate statistics on any table used for processing work items. ClaimCenter deletes any existing statistics on these tables whenever ClaimCenter updates statistics. You can override this behavior 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. ClaimCenter 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, ClaimCenter uses the database-wide statistics defined on the `databasestatistics` element.

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, ClaimCenter 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, ClaimCenter 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 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, ClaimCenter uses 75, the database default.

Notes:

- For performance reasons, ClaimCenter does not currently create a histogram on publicid columns. These columns are rarely, if ever, referenced in a WHERE clause.
- Also for performance reasons, ClaimCenter 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.

Purging Old Workflows and Workflow Logs

Each time ClaimCenter creates an activity, the activity is added to the cc_Workflow, cc_WorkflowLog and cc_WorkflowWorkItem tables. Once a user completes the activity, ClaimCenter sets the workflow status to completed. The cc_Workflow, cc_WorkflowLog and cc_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.

ClaimCenter 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 ClaimCenter/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 ClaimCenter/admin/bin directory with the following command:

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

Purging Orphaned Policies

Each claim has an associated policy record. If a user has refreshed the policy information on a claim, the old policy record remains in the database. If orphaned policies are consuming large amounts of database space, contact Guidewire Support for assistance removing the policies.

Purging Unwanted Claims

You can purge unwanted claims from the database. Unwanted claims could be claims that are older than a certain threshold, or they could be claims that were mistakenly or incorrectly entered into ClaimCenter. You can also store a date on the claim (`Claim.PurgeDate`) that records when the claim can be purged. ClaimCenter includes rules you can use to set the claim purge date when the claim is archived and to null the date if the claim is restored. See “Archive Claim Purge Rules” on page 51 in the *Rules Guide*.

Purging a claim permanently removes the claim from the database or the archive if archiving is enabled. You will never again be able to find the claim by searching, and you will never be able to restore it. Purging removes all traces of the claim from the database and the archive.

For ClaimCenter to purge a claim, the claim must meet all of these conditions:

- Unless it has draft status, it can not have any active (unanswered) messages, or ClaimCenter throws an `ActiveMessageException`.
- Unless it has draft status, it can not be part of a workflow, or ClaimCenter throws an `ActiveWorkflowException`.
- The claim cannot have aggregate limits.
- The claim cannot be linked to a bulk invoice.

Note: These conditions do not prevent you from purging an open claim.

As you purge a claim:

- All traces of the claim disappear from the main database. ClaimCenter uses the claim domain graph, which identifies all tables and rows associated with the claim, for purging.
- ClaimCenter removes the claim from all claim associations. If there is only one claim remaining in the association, ClaimCenter also removes the association itself.
- ClaimCenter adjusts all policies’ aggregate limits to remove all contributions made by the purged claim.
- If there are no policies associated with the claim’s policy period, ClaimCenter removes the policy period.
- If claims are held in an external database, ClaimCenter sends it an event message to request the purge of each claim.

Before doing a purge, back up the ClaimCenter database.

Purge claims either from the command line or by calling a SOAP API.

To use the SOAP API, call `IMaintenanceToolsAPI.startBatchProcess("process")` on the batch server, for example, `IMaintenanceToolsAPI.startBatchProcess("purge")`. See also “Web Services (SOAP)” on page 25 in the *Integration Guide* and “General Web Services” on page 81 in the *Integration Guide*.

To purge a claim from the command line

1. Use `maintenance_tools` to mark claims for purging.

   Mark a single claim or comma-delimited list of claims for purging by listing the claim number or numbers:

   ```
   maintenance_tools -password password -markclaimsforpurge -claims claim_number1, claim_number2...
   ```

Mark multiple claims by creating a text file containing a list of claim numbers separated by commas or new lines. Then mark them all for purging by using the following command:

```
maintenance_tools -password password -markclaimsforpurge -file file_name
```

After you mark claims for purging, you can no longer view or edit those claims. However, the claims still exist in the ClaimCenter database until you actually purge them. Upon marking a claim for purge, ClaimCenter retires the row in the `ClaimInfo` and `Claim` tables and deletes related rows that are not retireable from `LocationInfo`, `ClaimInfoAccess`, `ContactInfo`, and `PeriodPolicy`.

2. You can then purge the marked claims by using the following command:

```
maintenance_tools -password password –startprocess purge
```

ClaimCenter permanently deletes all claims that you marked for purging from the ClaimCenter database and the archive. ClaimCenter also deletes the `ClaimInfo` record for the claim.

Recalculating Financial Summaries

ClaimCenter calculates financial values such as open reserves, total payments, and gross check amounts. ClaimCenter stores these values in the `cc_claimrpt`, `cc_exposurerpt`, `cc_checkrpt` database tables. If any of the database consistency checks that ClaimCenter performs on these tables fails, instruct ClaimCenter to recalculate these values and repopulate the database tables. Use the following command:

```
maintenance_tools –password password –startprocess financialscalculations
```

Only run this process if you encounter consistency check failures. You can not schedule this process to run periodically.

Rebuilding Contact Associations

If any of the database consistency checks that ClaimCenter performs on claims and their contacts fails, instruct ClaimCenter to recalculate these associations and repopulate the database tables. Use the following command:

```
maintenance_tools –password password –startprocess claimcontactscalculations
```

Only run this process if you encounter consistency check failures. You can not schedule this process to run periodically.

Backing up the ClaimCenter Database

ClaimCenter 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 ClaimCenter database. You can take a hot backup while users are still accessing ClaimCenter. 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 ClaimCenter database from a backup, instruct ClaimCenter to rebuild its database statistics:

```
maintenance_tools –startprocess dbstatistics –password password
```

In addition to backing up the database, maintain a backup of the ClaimCenter configuration. This is especially important for any files that you modify as part of installation or configuration of ClaimCenter. All of these files are located within the `ClaimCenter/modules/configuration/config` directory, making it easy to keep those files in a source control system, if desired. See “Linking Studio to a SCM System” on page 91 in the *Configuration Guide*.

In the case of a complete system failure, with proper backups you can reinstall the ClaimCenter 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 ClaimCenter from the last database backup.

Resizing Columns

After the ClaimCenter database is in use, you might discover that you need to change the size of certain columns, such as making a column name longer. ClaimCenter 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 ClaimCenter.
2. Using the database tools, 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 ClaimCenter.

IMPORTANT Guidewire does not support resizing base columns.

chapter 5

Managing ClaimCenter Servers

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

This topic includes:

Stopping the ClaimCenter Application

Before you stop the ClaimCenter 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 ClaimCenter **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 ClaimCenter.
4. For any process that has a **Next Scheduled Run** time that is before the time that you will stop ClaimCenter, click **Stop** in the **Schedule** column. This disables the schedule for the current ClaimCenter session. The schedule is enabled again when you restart ClaimCenter, according to the settings in `scheduler-config.xml`. See "Scheduling Batch Processes and Distributed Work Queues" on page 143.

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

Understanding Server Run Levels and Modes

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

Production mode is not available for QuickStart servers. A QuickStart server always runs in development 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. Also see "Testing Clock Plugin (Only For Non-Production Servers)" on page 336 in the *Integration Guide*.
- ClaimCenter 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 ClaimCenter indicates that ClaimCenter 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 in the following table.

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	Archive 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.	Not allowed.

System Run Level	Simplified Run Level	Production mode	Development mode	Archive mode
DAEMONS	daemons	User interface not available. Web services available. Distributed Work Queue (including workflow) available. Workflow Stat Manager available. Scheduler available. Daemons started by application, such as QPlexor (for messaging) available.	User interface available. All logins allowed. Web services available. Distributed Work Queue (including workflow) available. Workflow Stat Manager available. Scheduler available. Daemons started by application, such as QPlexor (for messaging) available.	Not allowed.
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.	User interface available. All logins allowed. Web services available. No restriction on user.
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.	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.	User interface not available. Web services not available. Database not available.
NONE	Reported as starting	Nothing available.	Nothing available.	Nothing available.

Setting the Server Run Level

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

To set the server run level to multiuser

```
ClaimCenter/admin/bin/system_tools –password password –multiuser
```

To set the server run level to daemons

```
ClaimCenter/admin/bin/system_tools –password password –daemons
```

To set the server run level to maintenance

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

You cannot set server to the `SHUTDOWN`, `GUIDEWIRE_STARTUP` or `NONE` run level. However, `ISystemToolsAPI.getRunlevel()` can report these run levels.

If you run ClaimCenter 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 `ClaimCenter\admin\bin`. The following example shows how to determine the run level.

```
ClaimCenter/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 `ISystemToolsAPI.getRunlevel()`.

You can also determine the server run level from an unauthenticated web page. See “Checking Server Run Level” on page 89.

Using the Maintenance Run Level

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

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

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

ClaimCenter still allows connections made through ClaimCenter APIs or command line tools for any daemons with a minimum run level equal or lower than `NODAEMONS`. This permits integration processes to proceed without interference from unplanned activities by non-administrator users. See “Publishing a Web Service” on page 32 in the *Integration Guide*.

Understanding System Users

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

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

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

Graph Validation Checks

The ClaimCenter server performs a series of checks during startup. These checks verify that the current data model can support archiving, purging and export of claims. Some of these checks prevent the server from starting. Other checks are warnings and allow the server to start. These checks warn about potential problems with the graphs that might not be an issue depending on business logic.

The following table describes checks that prevent the server from starting. If any of these checks fails, the server reports an exception and prints out the cycle in DOT format. You can use a program such as GraphViz to view the graph.

Check	Description
Domain graph not partitioned	Verifies that all domain graph tables are reachable through the root entities.
Edge tables in domain graph have required foreign keys	Verifies that if `edgeTable` is set on an entity in the domain graph, it must have all of the following: • An owned foreign key back to one of its parents • An unowned foreign key to that same parent.
No cycles in domain graph	Looks for circular references in the domain graph. The domain graph is the set of tables and their relationships that define an archival unit, such as a Claim, its Exposures and Contacts, and so forth. The archiving and purging processes move non-shared data related to a Claim to the archive by traversing the domain graph. Circular references in the graph cause issues for this process. The domain graph cannot have any cycle in its “is owned by” relationships. Thus, the following example fails validation: A “is owned by” B “is owned by” C “is owned by” A. You need to resolve any cycles by sorting out the ownership relationships.
Domain graph entities implement `Extractable`	Ensures that all entities inside the domain graph implement the `Extractable` delegate as ClaimCenter uses the following columns on `Extractable` during the archive process: • `ArchivePartition` • `ArchiveID` This check also verifies that no entities outside of the domain graph implement the `Extractable` delegate.
Overlap tables implement `OverlapTable`	Verifies that all overlap tables implement the `OverlapTable` delegate. An overlap table contains rows that can exist either in the domain graph or as part of reference data, but not both. Overlap tables must implement the following delegates: • `Extractable` • `OverlapTable`
Entities in domain graph keyable	Verifies that all entities in the domain graph are `keyable`. This requirement enables you to reference the entity by ID.
Reference entities retirable	Verifies that all reference entities in the domain graph points are retireable.
Exceptions to links from outside the domain graph are outside the domain graph	Verifies that exceptions to links from outside the domain graph are actually from entities that are outside of the domain graph. ClaimCenter has several built-in exceptions to the general rule to not have a foreign key from an entity outside the domain graph to an entity inside the domain graph. For these entities, this check verifies that the outside entity is indeed outside the domain graph.

If any of these checks finds an issue, ClaimCenter prevents the server from starting and prints the graph in DOT notation. You can use the DOT format graph output to view the graph using graph visualization software such as GraphViz.

In addition, the **Archive Info** page provides warnings for situations that could potentially lead to errors. See “Domain Graph Info” on page 161 in the *System Administration Guide*. ClaimCenter provides warnings for these

situations rather than preventing the server from starting because business logic may prevent the erroneous situation. The following table describes these warning-level checks.

Check	Description
Nothing outside the domain graph points to the domain graph	There must not be foreign keys from entities outside of the domain graph to entities in the domain graph. This prevents foreign key violations as ClaimCenter traverses the domain graph. However, the existence of such foreign keys does not cause outright failure as it is possible for an archiving rule to prevent the archiving of such graph instances.
Null links cannot make node unreachable	ClaimCenter constructs the domain graph by looking at foreign keys. However, this can create a *disconnected* graph if a nullable foreign key is null. If enough links are null, the graph becomes partitioned and the archiving or purging process is not able to tag the correct entities. This check is a warning rather than one that prevents the server from starting as it is possible to use business logic to prevent the issue.

See also

- "Archiving" on page 87 in the *Application Guide* – information on archiving claims, searching for archived claims, and restoring archived claims.
- "Archive Parameters" on page 39 in the *Configuration Guide* – discussion on the configuration parameters used in claims archiving.
- "Archiving Claims" on page 665 in the *Configuration Guide* – information on configuring claims archiving, selecting claims for archiving, and archiving and the object (domain) graph.
- "Archiving Integration" on page 285 in the *Integration Guide* – describes the archiving integration flow, storage and retrieval integration, and the `IArchiveSource` plugin interface.
- "Archive" on page 48 in the *Rules Guide* – information on base configuration archive rules and their use in detecting archive events and managing the claims archive and restore process.
- "Logging Successfully Archived Claims" on page 37 in the *System Administration Guide*.
- "Purging Unwanted Claims" on page 58 in the *System Administration Guide*.
- "Archive Info" on page 160 in the *System Administration Guide*.
- "Upgrading Archived Entities" on page 62 in the *Upgrade Guide*.

IMPORTANT To increase performance, most customers find increased hardware more cost effective than archiving unless their volume exceeds one million claims or more. Guidewire strongly recommends that you contact Customer Support before implementing archiving to help your company with this analysis.

Monitoring the Servers

You can use an HTTP ping utility provided by Guidewire to check server status. See "Checking Node Health" on page 87.

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 ClaimCenter does not run out of disk space for logs. Archive and truncate system logs periodically to prevent the ClaimCenter 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 13 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 ClaimCenter 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

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

Monitor the environment's message traffic to ensure that the integration is running smoothly. This section discusses topics in monitoring and managing event messages. To learn about writing a message destination object, see "Messaging and Events" on page 139 in the *Integration Guide*.

How ClaimCenter Processes Messages

Every time ClaimCenter sends an event message, it expects to receive a positive acknowledgement (ack) back from the destination indicating it received and processed the message. ClaimCenter retains completed messages until you purge them. Since the number of messages in ClaimCenter 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 (*suspended*). Or, the destination is falling behind. ClaimCenter can generate messages very quickly
Needing Retry	Waiting to attempt a retry. ClaimCenter attempted to send the message but the destination threw an exception. If the exception was retryable, ClaimCenter automatically attempts to retry the send before turning the message into a failure. ClaimCenter 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	ClaimCenter 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.

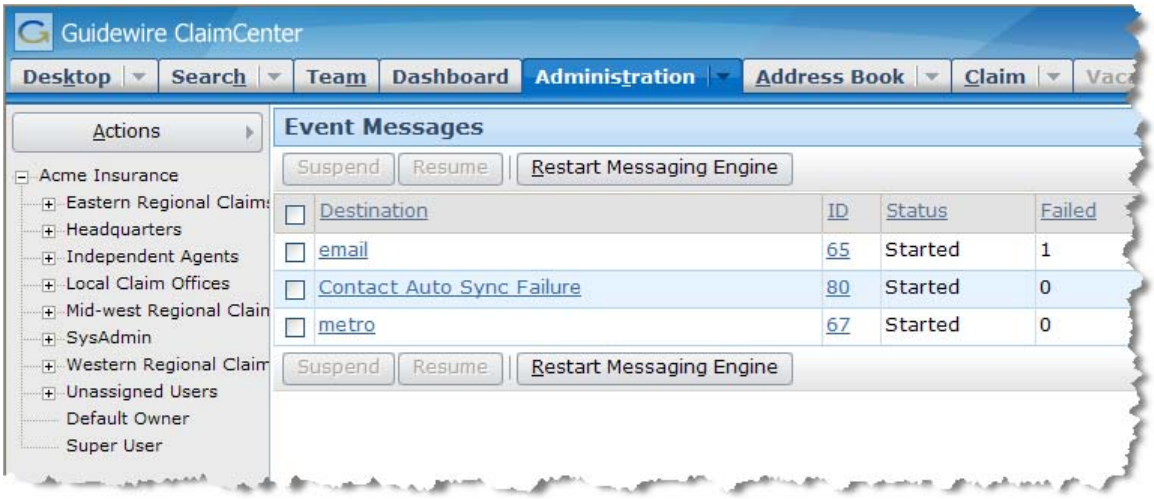
If ClaimCenter receives an unrecoverable or unexpected exception from a send attempt, or reaches the retry limit, it does not send messages to that destination until you clear the error. If ClaimCenter receives a processing error that is not retryable, ClaimCenter also suspends the destination and waits for you to clear the error. To clear

an error, either manually retry the send or skip the message. Do this from the user interface or from the command line.

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

Working with the Destinations Page

ClaimCenter lists each message destination in the **Event Messages** page. You access this page from the **Administrator** 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 ClaimCenter 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 a claim (for all destinations). You can search for a particular claim to respond to a query from an end user and then open the claim's detail view. From this page, you can select one or more claims and indicate that the failed or in-flight message for each claim be skipped, retried, or resynced.

Configuring Message Destinations

You create and configure message environments and destinations with the Guidewire Studio **Messaging** editor. See "The Messaging Editor" on page 161 in the *Configuration Guide*.

Tuning Message Handling

A ClaimCenter 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 ClaimCenter 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 ClaimCenter environment.

To address this situation, ClaimCenter provides the `IncrementalReaderSafetyMarginMillis` parameter in the `config.xml` file. This determines how long after detecting a skipped message that ClaimCenter 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 ClaimCenter assumes the message is lost and will never be committed, and ClaimCenter skips the message permanently.

For example, in the previous scenario, ClaimCenter 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, ClaimCenter 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.

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

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

You can configure the timeout value for a session by setting the value of the following property:

```
<security sessiontimeoutsecs="10800"/>
```

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

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

Avoiding Session Replication

ClaimCenter 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:

- ClaimCenter sessions are not serializable. Therefore, you cannot replicate a ClaimCenter session, either with or without persistence to the database.
- ClaimCenter 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.
- ClaimCenter commits changes to the database on almost all transactions. Noticeable exceptions are some wizards for which ClaimCenter commits data changes only after the user completes all necessary entries.
- ClaimCenter 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 ClaimCenter with a single sign-on solution to prevent users from having to log back in at failover.

Understanding 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, ClaimCenter 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, ClaimCenter never evicts the cache entry, but the entry does eventually become stale. You can set the default evict time by adjusting the `GlobalCacheReapingTimeMinutes` parameter in `config.xml`. `GlobalCacheReapingTimeMinutes` is initially set to `15` minutes. The minimum value for this parameter is `1` minute. The maximum value for this parameter is the smaller of `120` and `GlobalCacheStaleTimeMinutes`.

- Upon reception of an inter-cluster message indicating that an object value was changed in another node, the server marks the corresponding entry in the cache obsolete and available for reuse.

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

Caching and Stickiness

The cache mechanism is fully leveraged if users return to the same node across different HTTP requests. In a clustered environment, the load balancer must direct requests to the same application server node upon consecutive interactions. This mechanism, referred to as “stickiness”, enables a true horizontal scalability solution. For more information on load balancing options for ClaimCenter, 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 ClaimCenter updates an object value to the database, ClaimCenter compares a counter associated to the object to the counter in the database. A counter value mismatch indicates that the object was concurrently changed. In such case, ClaimCenter rejects the change and the cache mechanism throws a concurrent change exception. ClaimCenter presents the user who initiated the concurrent change with the error and reloads the latest data. The user can then reapply the changes. Furthermore, ClaimCenter commits changes in an atomic bundle, ensuring transactional integrity. Therefore, ClaimCenter 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, ClaimCenter uses the optimistic locking mechanism.

Caching and Clustering

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

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

Performance Impact

Guidewire gathered some additional information on the performance order of magnitudes regarding caching.

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.

For an action involving approximately 2000 objects, Guidewire gathered the following metrics:

Action	Time
Process data	2 seconds
Load data from the database cache to the application cache	4-5 seconds
Load data from storage to database cache	15-18 seconds

Therefore, the same action takes 2 seconds if the cache contains the data and up to 25 seconds if the data is neither in the application server or database cache.

Proper caching behavior is critical to performance. “Analyzing and Tuning the Application Server Cache” on page 73 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 ClaimCenter user works on a claim, ClaimCenter 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 74.

Cache Impact on Memory Utilization

The maximum size of the cache is dependant on cache parameters. See “Analyzing and Tuning the Application Server Cache” on page 73. 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 75.

Analyzing and Tuning the Application Server Cache

The `config.xml` file contains cache parameters for ClaimCenter. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**.

Parameter	Description
`CacheActive`	Boolean indicating whether to use the cache. Controls whether the cache is active. If you set this value to false, then you effectively set the maximum allowable space for the global cache, `GlobalCacheSizeMegabytes`, to `0`. `CacheActive` affects the global cache, used for most objects. There are also special caches for rarely changing objects. These include the following: • Exchange rates • Groups • Script parameters • Zones The `CacheActive` parameter does not affect these special caches. For more information about the special caches, see “Special Caches for Rarely Changing Objects” on page 75.
`GlobalCacheSizePercent`	Maximum amount of heap space used to store cached entities, expressed as a percentage of the maximum heap size.
`GlobalCacheSizeMegabytes`	Maximum amount of heap space used to store cached entities, expressed as a number of megabytes. This parameter supersedes the value of `GlobalCacheSizePercent`.
`GlobalCacheStaleTimeMinutes`	Time, in minutes, after which ClaimCenter 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, ClaimCenter prefers this mechanism over evicting objects upon stale timeout.
`GlobalCacheReapingTimeMinutes`	Time, in minutes, since last use of a cached object before ClaimCenter considers the object eligible for reaping. This can be thought of as the period during which ClaimCenter 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, ClaimCenter 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.

Parameter	Description
GlobalCacheActiveTimeMinutes	Time, in minutes, that ClaimCenter 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.
GlobalCacheStatsWindowMinutes	This parameter denotes a period of time, in minutes, that ClaimCenter uses for two purposes: • how long to preserve the reason that ClaimCenter evicted an object, after the event occurred. When a cache miss occurs, ClaimCenter 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 158.
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 75.
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 75.
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 75.
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 75.

Analyzing Cache Settings

See "Cache Info" on page 158 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 dependant 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 ClaimCenter 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, ClaimCenter includes caches specific to rarely changing objects. ClaimCenter 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 ClaimCenter 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 73.

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.

ClaimCenter 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 76.

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 using 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.

Using Tools to Analyze Garbage Collector Behavior

Verify that the performance analysis tool you choose supports the version of the JVM that you use for ClaimCenter. For supported JVM versions, see “Installing Java” on page 27 in the *Installation Guide* and “Configuring the Application Server” on page 13 in the *Installation Guide*.

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-01.ibm.com/software/support/isa/` for more information about the IBM Support Assistant.

HPjmeter and GCViewer

HPjmeter and GCViewerare 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 using 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://docs.hp.com/en/5992-5899/index.html`

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

Analyzing a Possible Memory Leak

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

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

Investigation of memory leaks differs slightly per JVM platform.

Common approach

Various options exist to generate a heap dump:

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

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

Generating Heap Dumps with IBM JDK

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

Generating Heap Dumps with Oracle HotSpot JDK

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

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

Additional Recommendations

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

- Heap dump generation frequently fails because the single file generated is very large and the supporting environment is configured to prevent regular accounts to create such large files. Therefore, some configuration is generally required to allow the account running the node to create such large files.
- The generation of a heap dump during out-of-memory conditions is sometimes challenging. As a JVM is reaching maximum memory utilization, it generally experiences severely degraded performance. As the pace of the leak decreases gradually, the occurrence of the out-of-memory condition might take an inordinate amount of time. This length of time might be incompatible with the need to restore performance for users or processes.
- Windows only: Windows does not support signals. Therefore, generating a heap by starting the JVM with a heap dump on `CTRL-BREAK`, depends on the capacity to send a `CTRL-BREAK`. You cannot send a a `CTRL-BREAK` to a JVM started as a background process. Therefore, for the time of the investigation, start the JVM from a command line rather than as a background process.
- The JVM generally provides optional flags that prevent it from listening to signals. Disable these flags while trying to generate a heap dump through signals.
- Heap dump analysis is very memory intensive. Assume that the tool used to analyze the heap dump might need a heap two to three times larger than the amount of objects captured in the heap dump. Host the heap dump analyzer on a server with a 64-bit JVM and a significant amount of memory. If such a configuration is not available, you might want to reduce the heap size so that the memory leak reaches an identifiable threshold sooner. This method allows the generation of smaller, easier to analyze heap dumps.
- Heap dump analysis tools generally point to the `CacheImpl` class as the largest memory consumer. This class corresponds to the Guidewire cache. It is normal that the cache consumes a few hundred megabytes. In this case, the memory issue is likely not caused by cache growth. If the cache consumes significantly more memory, you might need to be downsize the cache. See “Understanding Application Server Caching” on page 70.

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-306.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 Heap Analysis Tool (HAT) with Java HotSpot VM 1.6. Therefore, if you want to analyze a heap with HAT, you can install Java HotSpot VM 1.6 and use the HAT release provided. Refer to the article *Heap Analysis Tool* at `https://hat.dev.java.net/` for more information.
- jVisualVM is another multi-purpose tool that you can use to analyze heap dumps. Refer to *JVisualVM* at `http://java.sun.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.

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

Managing Clustered Servers

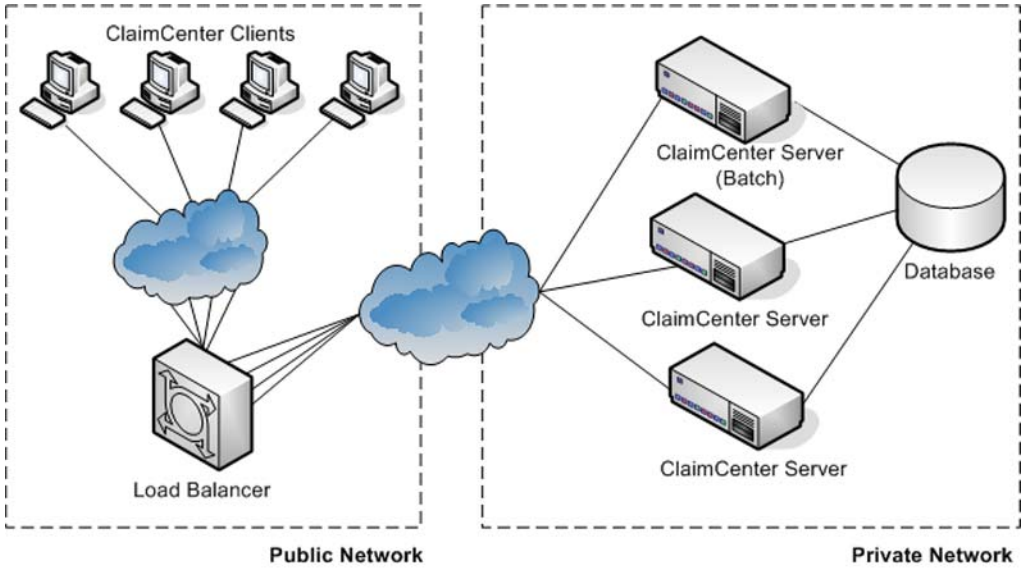
To improve performance and reliability, you can install multiple ClaimCenter servers in a configuration known as a cluster. A cluster distributes client connections among multiple ClaimCenter servers, reducing the load on any one server. If one server fails, the other servers seamlessly handle its traffic.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

Also review “Considerations for a Clustered Application Server Environment” on page 14 in the *Installation Guide*.

This topic includes:

Overview of Clustering

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



Plan the cluster so that if any one server fails, the other servers in the cluster can handle its traffic without being overwhelmed. ClaimCenter 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 ClaimCenter 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 ClaimCenter server. The load balancer must implement *session affinity*, meaning that it must route connections from the same user session to the same ClaimCenter server. If the load balancer directed a user to a different server, the session is reset. This can result in loss of unsaved data.

Within any cluster, there can be only one ClaimCenter batch server. The batch server acts as a typical ClaimCenter 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.

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

In general, start the batch server first. If the batch server goes down, either restart the batch server, or use the Management Beans section of Server Tools to designate another node as the batch server. See “Management Beans” on page 154. 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 ClaimCenter Batch Servers

ClaimCenter 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 ClaimCenter 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 ClaimCenter server. These tools cannot connect through a load balancing virtual host.
- Guidewire applications are application server independent. For this reason, if you implement a ClaimCenter cluster, Guidewire recommends that you not use proprietary application server features for failover, sharing sessions between nodes, and so forth. Instead, disable these features.

Configuring a Cluster

You install ClaimCenter on all the servers in a cluster in the same way that you install a standalone ClaimCenter server. Then, you create the cluster by changing each server's configuration settings. With some exceptions, all the servers in a cluster must have identical `config.xml` files and identical metadata. There are directories within the `configuration` module (`ClaimCenter/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.
`config/profiler`	Directories for putting extension code for the integration interfaces. See the *ClaimCenter Integration Guide* for more details.

As each server starts, it connects to another server in the cluster and compares its configuration environments. If the configurations differ, the server fails to start. To configure a cluster, you use several parameters in the

config.xml file and make use of the ClaimCenter environment properties to ensure that server-specific properties resolve correctly. See the following sections:

- “Enabling and Disabling Clustering” on page 84
- “Configuring the Registry Element for Clustering” on page 84
- “Setting the Multicast Address” on page 86
- “Specifying the Key Range” on page 86
- “Configuring Separate Logging Environments” on page 86

Before continuing to configure the cluster, review “Defining the Application Server Environment” on page 18.

Enabling and Disabling Clustering

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

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

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

Configuring the Registry Element for Clustering

The registry element and ClaimCenter environment properties play an important role in creating a cluster. Since the config.xml file and the config subdirectories must be identical, create a configuration that runs on all servers. If you have not already done so, review “Defining the Application Server Environment” on page 18. 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 18 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 ClaimCenter 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, ClaimCenter first resolves the env element. Assuming that -Dgw.cc.env is not set on the command line to another value, this registry would result in the env resolving to cluster1.

This same configuration can be used to start any of these servers in standalone mode. To do this, start the application server with a `-Dgw.cc.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 21.

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 ClaimCenter 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. If you configure multiple servers in a cluster to act as a batch server, then the servers race for the designation. The first server that starts becomes the batch server — the others operate as regular servers regardless of their designation. In a standalone configuration in which `ClusteringEnabled` is `false`, the individual server always acts as its own batch server.

Identify a 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, ClaimCenter 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.cc.isbatchserver` option:

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

For a complete discussion of how environment properties work, see “Defining the Application Server Environment” on page 18.

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

1. ClaimCenter 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, ClaimCenter sets `isbatchserver` to `false` by default.

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

Setting the Multicast Address

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

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

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

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

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

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

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

224.0.0.0 – 239.255.255.255

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

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

Specifying the Key Range

When you create a new ClaimCenter object such as a Claim, ClaimCenter assigns it a key, or unique public identifier. To ensure that keys are unique, the server requests an available key from the ClaimCenter 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. There are about two billion keys available, so wasting a few is not much of 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 `ClaimCenter/modules/configuration/config/logging` directory on each server. If that file does not exist or the `env` property does not resolve to `cluster1`, ClaimCenter 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 36.

Managing a Cluster

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

Starting Clustered Servers

To start ClaimCenter 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 ***** ClaimCenter ready *****
   ```

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

   ```
   http://server:port/cc/ping
   ```

   When the server is running at the default `MULTIUSER` run level, the browser displays the number `2`. For more information on the ClaimCenter HTTP ping utility, see “Checking Node Health” on page 87.
5. For each web service server, invoke `http://server:port/cc/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 ClaimCenter servers for handling user requests. After executing each server start, wait ten seconds. Then, start the next ClaimCenter server.

   The last step assumes that the ClaimCenter 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 ClaimCenter servers. Instead, just wait for ten seconds after the previous server startup script is executed.

Checking Node Health

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

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

Some other infrastructure constraints exist. For example, some environments cannot use source IP stickiness to load-balance conversational SOAP calls. In this situation, administrators generally instantiate one ContactManager instance on each server hosting ClaimCenter instances and configure ClaimCenter to direct contact requests to that local ContactManager instance. In this case, if that ContactManager instance is not functioning properly, it is advisable to stop directing traffic to any of the ClaimCenter instances on that server.

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

```
http://localhost:8080/cc/ping
```

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

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

Guidewire based this HTTP ping utility on a system run level checker built for developers. See “Getting and Setting the Run Level” on page 85 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. For example, you could have an environment in which ClaimCenter directs traffic to a local ContactManager instance. You could configure the load balancer to only redirect traffic to the ClaimCenter node if both that node and the ContactManager instance on that server are accessible for user requests.

Adding a Server to a Cluster

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

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

To add a server to the cluster

1. On the server you want to add, remove the `config` directory and the `config.xml` file.
2. Copy the `config` directory and the `config.xml` file from a server on the cluster to the server you want to add.
3. If the new server is on the same physical machine as another server, set the `serverid` system property in the `config.xml` file to a unique value within the cluster. By default, the `serverid` defaults to the hostname of the machine running the instance. This works well in most configurations but causes problems if multiple instances are running on a single machine. See “Defining the Application Server Environment” on page 18.
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. ClaimCenter writes failure messages to the log file:

```
-------------------------------------------------------
GMS: address is havasu:3491
-------------------------------------------------------
```

```
st:8085/cc  2006-05-05 14:16:02,963 INFO Cluster channel started
st:8085/cc  2006-05-05 14:16:02,963 INFO Starting clustered inittab
st:8085/cc  2006-05-05 14:16:02,963 INFO Started clustered inittab
st:8085/cc  2006-05-05 14:16:02,978 INFO Starting clustered config verifier
st:8085/cc  2006-05-05 14:16:13,979 ERROR Local server configuration doesn't match configuration for
  servecdcdr havasu:3476
st:8085/cc  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/cc  2006-05-05 14:16:13,994 ERROR
  An exception was thrown while starting a component. Setting runlevel to
  SHUTDOWN [Configuration mismatches]
com.guidewire.GWLifecycleException: Configuration mismatches at
  com.guidewire.pl.system.cluster.ClusteringComponent.start(ClusteringComponent.java:153)
```

Checking Server Run Level

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

```
http://server:port/cc/ping
```

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

Code	Run level
2	MULTIUSER
-	DAEMONS
(	MAINTENANCE

Server Failures and Removing a Server

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

To remove a server from the cluster

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

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

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

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

With this configuration, the server that starts first, becomes the batch server. In the event that one of the servers does not start, you still have a batch server without needing to reconfigure and redeploy a new configuration across the cluster. Instead, simply restart the appropriate computer. Or, use clustering software to provide high availability for the batch server.

If you remove 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. You must then start another 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 ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

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

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

chapter 7

Enabling JMX with ClaimCenter

JMX (Java Management Extensions) let you configure and monitor ClaimCenter remotely.

This topic includes:

Overview of JMX Management Tasks

ClaimCenter provides access to management features through JMX (Java Management Extensions), letting you configure certain settings and monitor various aspects of ClaimCenter operation. After you enable JMX, you can use a JMX management console to connect to the ClaimCenter JMX server.

Through JMX you can do the following:

- **Configure –** View the values of all configuration parameters. Also, you can change the value of certain parameters. These changes take effect immediately without requiring you to restart the server. Changes that you make through JMX are discarded when the server shuts down. Upon restart, the server uses parameter values that are specified in the `config.xml` configuration file.
- **Monitor –** You can view the following system information:
  - batch processes currently running, if any
  - number of current user sessions and their user names
  - number of active and idle database connections

  You can also monitor notifications, which are triggered when ClaimCenter locks a user out for excessive login failures.

ClaimCenter uses the MX4J 1.1 implementation of JMX. For a list of the `MBeans` available with ClaimCenter, see "Management Beans" on page 154.

Preparing the Management Plugin

Before you enable JMX in ClaimCenter, you must prepare the Java classes that implement the management plugin.

To prepare the Management plugin

1. Run the command `gwcc regen-java-api` from the `ClaimCenter/bin` directory.
2. In your Java development environment, compile the `.java` files in the `ClaimCenter/java-api/examples/src/examples/plugins/management` directory.
3. Do one of the following:
   a. Create a `management/classes` directory in the `ClaimCenter/modules/configuration/plugins` directory. Then, move the compiled `.class` files to the `management/classes` directory that you just created, including the `examples/plugins/management` package directories.
   b. Create a `management/lib` directory in the `ClaimCenter/modules/configuration/plugins` directory. Then, put the compiled `.class` files in a JAR file, including the `examples/plugins/management` package directories, and put the JAR file in the `management/lib` directory that you just created.

Result

The management plugin is prepared and installed in the correct location within your `ClaimCenter` directory. You are ready to enable JMX in `ClaimCenter`.

Enabling JMX in ClaimCenter

ClaimCenter provides a management plugin that you can use for JMX connections. Before you begin this procedure, you must compile the Java classes that implement the management plugin. For more information, see “Preparing the Management Plugin” on page 92

By default, JMX connections use port 1099. If that port is in use already in your installation of ClaimCenter, you can change the port that JMX uses to another, unused port number.

To enable the Management plugin

1. Open Guidewire Studio by running `gwcc studio` from the `ClaimCenter/bin` directory.
2. In the **Resources** pane, select **Plugins** → **gw** → **plugin** → **management** → **ManagementPlugin**.
3. Select the **Enabled** checkbox. Studio opens a dialog box informing you that editing the plugin creates a copy in the current module.
4. Click **Yes** on the dialog box.
5. If you want to use a port for the management server other than the default of 1099, edit the value for the *`rmiPort`* parameter.
6. Click **File** → **Save Changes**.
7. Do one of the following:
   c. If you are enabling the plugin in a QuickStart configuration, restart the ClaimCenter server by running `gwcc dev-start` from `ClaimCenter/bin`.
   d. If you are enabling the plugin in a production environment, repackage the ClaimCenter `.ear` or `.war` file, and redeploy the file to the application server.

For more information, see “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Regardless of configuration or environment, the server starts correctly if on startup the console log shows something similar to the following:

```
Starting JSR 160 connector on port 1099
```

8. In the JMX management console, create a JMX server connection using the following settings:

Setting	Value
`host`	The `hostname` of the ClaimCenter server.
`JNDI name`	jrmp
`port`	`1099` (default), or the port number that you configured in Studio

IMPORTANT Check that the port you assign for the JMX server connection is not already in use. If the port is in use, specify an available port.

chapter 8

Securing ClaimCenter Communications

Guidewire products use a standard three-tier architecture:

- The browser tier presents the ClaimCenter 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 and web/application server to a level strong enough that it cannot be easily compromised. This section describes what is required, in the simplest sense, 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 ClaimCenter over SSL (secure socket layer) for at least the login and change password pages. Ideally, deploy ClaimCenter entirely under SSL to protect all sensitive transmitted data.

This topic includes:

Using SSL with ClaimCenter

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 ClaimCenter application server using 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. “Modifying the httpd.conf File” on page 96
2. “Editing the httpd-ssl.conf File” on page 96
3. “Modifying the server.xml File” on page 97

Modifying the httpd.conf File

You configure the Apache server 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 cc6.0.8 and
      #its subdirectories for the authorized sites only
      <Directory /cc6.0.8>
            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           /cc http://MyClaimCenterHost:8080/cc
      ProxyPassReverse    /cc http://MyClaimCenterHost:8080/cc
#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.

Modifying the server.xml File

When ClaimCenter 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 ClaimCenter 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 ClaimCenter Server Using SSL

Use a different address to access ClaimCenter using SSL. The new address resembles the following:

```
https://server:port/cc/ClaimCenter.do
```

Notice the use of the `https` protocol instead of `http`, indicating that the server is connecting using 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, ClaimCenter plugins, and applications using the ClaimCenter APIs. Also, check the `config.xml` for any URL specifications that you need to change.

Handling Browser Security Warnings

When users connect to ClaimCenter over the SSL connection, they might see a security warning stating that they are connecting to a secure server. Users can simply click **Yes** to proceed and connect to ClaimCenter. 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 **Internet** or **Intranet** whichever is appropriate.
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 9

Securing Access to ClaimCenter Objects

Guidewire designed its security infrastructure to enable you to add custom permissions, automatically enforce permissions, and easily map between users permissions and actions. This topic explains how to use the permission infrastructure to control access to key ClaimCenter objects.

This topic includes:

Understanding the Object Access Infrastructure

You can assign each user one or more roles that contain permissions. These permissions control what the user can do within the system. For example, a user with the **Adjuster** role can create a claim. The limitation of roles is that they do not distinguish among objects of the same type. Claim, in this example, means all claims. ClaimCenter provides an access control feature that allows you to overcome this limitation of roles.

Using access control, you can subcategorize an object type and then restrict object access by these subcategories. You can apply access control to claim, exposure, and document objects. This topic introduces the kinds of permissions within ClaimCenter and explains how permissions interact with access control.

This topic includes:

- Understanding the Different Permission Types
- The Security Dictionary

- Adding a System Permission
- Beyond Roles and Permissions to Access Control

Understanding the Different Permission Types

ClaimCenter contains individual system permissions and *application permission keys*. Application permission keys represent a set of one or more system permissions. System permissions fall into two categories: *screen-level permissions* and *domain-level permissions*. Screen-level permissions apply to user interface elements, such as permission to view a particular the claim summary screen. Domain permissions apply to data model entities, such as permission to view `Claim` objects. When a user attempts to access the claim summary for a sensitive claim, ClaimCenter verifies that the user has permission to view both the **Claim** screen and that claim's data.

Most top-level objects in the ClaimCenter data model have domain-level permissions. ClaimCenter defines all of an object's domain-level permissions internally; you cannot add, remove, or edit domain permissions. Similarly, the points at which these permissions are checked are defined internally within ClaimCenter code and externally in the page configuration format (PCF) files. You cannot change the internal checks but you can change when they are called in the PCF files.

Similarly, ClaimCenter defines many user interface permissions internally. In general, screen-level permissions start with the word "view" followed by a reference to the user interface object they protect. You *can add* custom screen-level permissions to the system. PCF files define when ClaimCenter calls user interface permissions. You can change when ClaimCenter calls a permission by customizing PCF files.

Application permission keys, as stated previously, represent sets of system permission keys. ClaimCenter defines application permission keys internally as a method for improving system performance. For example, the `claimedit` permission key represents the system permissions for the editing a claim: `claimedit` for open claims and `claimeditclsd` for closed claims. Application permission keys do appear in the PCF files, so it is important to know what they are. However, you cannot create or modify application permission keys.

The Security Dictionary

You can use the *Security Dictionary* to view all the system permissions and application permission keys. Using the dictionary, you can determine:

- when ClaimCenter relies on a particular permission key and the system permission keys that it references
- the permission keys, PCF files, or roles that reference an individual system permission
- a list of the permissions called from each PCF page
- a list of the permissions available for each role

To build the *Security Dictionary*, run the following command from `ClaimCenter\bin`:

```
gwcc regen-dictionary
```

This command builds the *Security Dictionary* in the following location:

```
ClaimCenter\build\dictionary\security\index.html
```

Adding a System Permission

Edit the `SystemPermissionType` typelist to add your own screen-level permissions to ClaimCenter. You can also retire or modify existing permissions in this typelist. This typelist does not contain all the permissions in the system. Internal system permissions, both domain and user interface, do not appear in this typelist.

To add a permission

1. Open a command window to `ClaimCenter\bin`.
2. Execute the following command to launch Studio:

```
gwcc studio
```

3. In the Studio **Resources** pane, select **configuration** → **Typelists** → **SystemPermissionType**.
4. On the **Codes** tab of the **SystemPermissionType** pane, click **Add**.
5. Enter a **Code** for the new permission. Include the type of action you want the permission to control in the code, for example, `viewvacation`. The **Code** must be 16 characters or less.
6. Add a **Name** and **Description** to the new permission. Leave the **Priority** as `-1` and **Retired** as `false`.
7. Select **File** → **Save Changes**.
8. Rebuild and redeploy your application package file. If you are in a test environment, you can just copy your configuration. In both cases, run `regen-dictionary` to update the typelist documentation.

Beyond Roles and Permissions to Access Control

You can group system permissions by adding them to roles and then assigning the role to a user. So, if the **Adjuster** role had create permission on exposure objects any user with that role can create one. In practice, you likely do not want all users to access all objects of the same type. For example, if Oprah Winfrey submitted a claim you might want to ensure that only very discrete adjusters have access to Oprah's personal information. The ClaimCenter access control feature allows you to make distinctions among objects of the same type and then secure access to them.

While roles and permissions determine *what* actions a user can perform, access control determines *on which* objects the user can act. When you enable access control, a user requires both the correct role and the proper access. To use access control you apply a security attribute to an object and then profile the users who have access to objects with that attribute. Within ClaimCenter you can use access control on claims, exposures and documents. Though, the access control on exposures and documents is somewhat less sophisticated compared to that of claims.

Claim Access Control

Claim access control is the most sophisticated, which is proper given that claims are key objects. To use access control with claims, you must:

- Specify the claim security types. For example, you might need to be able to designate claims that come from celebrities as sensitive claims.
- Map the claim access to permissions. This specifies what you can do to a claim if you have a specific type of access. For example, with edit access a user can close a claim.
- Create an *access profile* for each claim security level. Within the profile, you specify *access levels* that define the organizational relationship to the claim a user must have to have a specific access type. For example, only the claim owner can edit the claim.

Using access profiles, you can restrict access by ownership, group, and security zone — or open access to anyone at all. You could specify that to view an unsecured claim a user must belong to the group to which the claim is assigned, but only the owner can modify the claim. You could also specify that, to edit claims that are at the sensitive level, a user must have a role that contains the `ownsensclaim` permission.

> **Note:** Access control is cumulative. If you grant access to a user at the group level, you can not remove that access by more restrictive access at the role level.

Having access is not the same as having permission. Even though an access profile may grant a user access to an object, that user still requires a role with the proper permissions. For example, suppose an access profile grants view access to any user who is a member of the claim owner's security zone. To view the claim, users must both belong to the proper security zone **and** have the `viewclaim` permission on one of their roles.

ClaimCenter supports *downline access* for supervisors. This feature effectively gives supervisors the equivalent access as any user, group, or security zone that they administer. This means, if a user has access to a claim, that user's supervisor also has access. Keep in mind, the supervisor must also have a role that grants the proper claim

permissions as well. You can change this default behavior by changing the access control parameters in the `config.xml` file.

Document and Exposure Access Control

You can use document and exposure permissions access control as well. The access control on documents and exposures not as extensive as claim access controls. For different classes of documents, you can create a mapping between document actions and permissions similar to the one you create for claims. For exposures, you can also create a action-permission mapping.

Note: Unlike claims, documents and exposure access controls do not support downline permissions.

Key Access Control Configuration Files

By default, ClaimCenter uses access control. You can change this configuration using the following `config.xml` parameters:

Parameter	Description
`UseACLPermissions`	Specifies whether ClaimCenter uses access control for claims at all. If set to `false`, then ClaimCenter ignores the access control definitions, and all users have access to all claims. ClaimCenter still restricts the actions that a user can take by the user's roles and permissions. This value only applies to claim access control. If you set this value to `false`, you can still use exposure and document access control features.
`RestrictSearchesToPermittedItems`	If set to `true`, then searches return only items that the user is able to view. If set to `false`, then search results might include items that the user cannot view.
`EnableDownlinePermissions`	If set to `true`, then a user has access to all claims to which any user or group they supervise or manage also has access. This applies recursively down the group hierarchy. If set to `false`, then a supervisor must have access that is independent of the users they supervise. This value only applies to claim access control.

Manage access control in ClaimCenter using the following files:

File	Role it plays in access control
`config.xml`	Contains parameters for turning access control on or off and for configuring how it behaves.
`/security/security-config.xml`	Defines the access control list elements.
`/extensions/ClaimAccessType.ttx`	Specifies different types of access related to claims.
`/extensions/ClaimSecurityType.ttx`	Defines the claim security types. This typelist specifies the contents of the claim user interface **Special Claim Permission** drop down.
`/extensions/DocumentSecurityType.ttx`	Defines document types for use with access control.
`/extensions/ExposureSecurityType.ttx`	Defines access control types related to exposures.
`/extensions/SystemPermissionType.ttx`	Contains customizable and custom system permissions.
`/extensions/UserRole.ttx`	Defines possible claim user roles. These roles appear on the **Users** tab of the claim.

You can modify these files using Guidewire Studio. The `config.xml` and `security-config.xml` files are available from the **Resources** pane under **configuration** → **Other Resources**. The typelist files are available under **configuration** → **Typelists**.

Controlling Access to Claim Objects

In this section, you learn how to use access control to refine a user's access to specific claim types. This section contains the following topics:

- Specifying Security Types
- Mapping Access Types to Permissions
- Creating Claim Access Profiles
- Examples of Claim Access Profiles

Specifying Security Types

The first step in establishing claim access control is to define claim security types. The `ClaimSecurityType` typelist defines these security levels. The security levels appear in the claim details under the **Special Claim Permission** drop-down.

Guidewire defines a number of default claim security types.

You can also add custom types to the `ClaimSecurityType` typelist using Studio.

To add a claim security type

1. Open a command window to `ClaimCenter\bin`.
2. Execute the following command to launch Studio:
   ```
   gwcc studio
   ```
3. In the Studio **Resources** pane, select **configuration** → **Typelists** → **ClaimSecurityType**.
4. On the **Codes** tab of the **ClaimSecurityType** pane, click **Add**.
5. Enter a **Code** for the new security type. Typically, the code identifies the special claim type, such as `employeeclaim`. The **Code** must be 16 characters or less.
6. Add a **Name** and **Description** to the new security type. Leave the **Priority** as `-1` and **Retired** as `false`.
7. Select **File** → **Save Changes**.
8. Rebuild and redeploy your application package file. If you are in a test environment, you can just copy your configuration. In both cases, run `regen-dictionary` to update the typelist documentation.

Notes:

- ClaimCenter assigns claims without a defined security type to the default type of `unsecuredclaim`. This is an internally-defined type and does not appear in the `ClaimSecurityType` typelist.
- You can also use Gosu to apply a security type to a claim. See the *ClaimCenter Rules Guide* for more information.

Mapping Access Types to Permissions

The second step in defining access control is to map claim access types to permissions. Claim access types group claim-related permissions. The mapping of claim access types to permissions determines what actions ClaimCenter permits a user to take within the scope of a given access type. For example, the `view` access type might consist of all of the permissions for viewing claim data, such as **View claim**, **View exposures**, **View activities**, and so forth.

To map access types to permissions

1. Open a command window to `ClaimCenter\bin`.

2. Execute the following command to launch Studio:

```
gwcc studio
```

3. In the Studio **Resources** pane, select **configuration** → **Other Resources** → **security-config.xml**.

   The `AccessMapping` element in the `security-config.xml` file specifies the map. This element has the following syntax:

```
<AccessMapping claimAccessType="access_type" systemPermission="perm" />
```

   The *access_type* is a claim access type. The *perm* is a claim-related system permission code. See "The Security Dictionary" on page 100 for information on listing available system permission codes. The following example maps the **Permission to create new activities on a claim** (`actcreate`) to the `create` access type:

```
<AccessMapping claimAccessType="create" systemPermission="actcreate" />
```

   Typically, multiple permissions are mapped to an access type. This is illustrated by the default `edit` access*:*

```
<AccessMapping claimAccessType="edit" systemPermission="actcreate"/>
<AccessMapping claimAccessType="edit" systemPermission="actcreateclsd"/>
<AccessMapping claimAccessType="edit" systemPermission="checktransfer"/>
<AccessMapping claimAccessType="edit" systemPermission="claimclose"/>
<AccessMapping claimAccessType="edit" systemPermission="claimedit"/>
...
```

4. Add an access mapping element, specifying a claim access type and a system permission.

5. Select **File** → **Save Changes**.

ClaimCenter does not require you to map all claim access types. However, access control does not function properly if you do not map them all. Mappings must meet the following requirements:

- You must map all claim-related system permissions.
- You can map a claim-related permission *only* to one claim access type. For example, you can map `paycreate` to `view` but not to `edit` also.
- Of the internal system permissions, you can only map *claim-related* permissions. For example, the `paycreate` permission allows a user to create a payment on a claim. It is claim-related. The `ruleadmin` permission is not claim related. Mapping it causes a configuration error.

If you add a custom permission, you can map that permission in the `security-config.xml` file.

The `ClaimAccessType` typelist defines claim access types. You can modify the default types, and you can define custom types.

To add a claim access type

1. Open a command window to `ClaimCenter\bin`.

2. Execute the following command to launch Studio:

```
gwcc studio
```

3. In the Studio **Resources** pane, select **configuration** → **Typelists** → **ClaimAccessType**.

4. On the **Codes** tab of the **ClaimAccessType** pane, click **Add**.

5. Enter a **Code** for the new access type. The **Code** must be 16 characters or less.

6. Add a **Name** and **Description** to the new security type. Leave the **Priority** as `-1` and **Retired** as `false`.

7. Select **File** → **Save Changes**.

8. Rebuild and redeploy your application package file. If you are in a test environment, you can just copy your configuration. In both cases, run `regen-dictionary` to update the typelist documentation.

IMPORTANT For performance reasons, Guidewire does not recommend that you define more than a few custom types.

Creating Claim Access Profiles

The final step to establishing claim access control is to create an access profile for each claim security level using the `AccessProfile` element. Using an access profile, you can control security related to the details, activities, and exposures of a claim.

The `AccessProfile` element is in the `security-config.xml` file and has the following syntax:

```
<AccessProfile securitylevel="level">
  <ClaimOwnPermission permission="perm"/>
  <SubObjectOwnPermission permission="perm"/>
  <ClaimAccessLevels>
    <AccessLevel level="level" permission="access_type" />
    <DraftClaimAccessLevel level="level"/>
    <ClaimUserAccessLevel role="user_role" level="level" permission="access_type" />
  </ClaimAccessLevels>
  <ActivityAccessLevels>
    <AccessLevel level="level" permission="access_type" />
  </ActivityAccessLevels>
  <ExposureAccessLevels>
     <AccessLevel level="level" permission="access_type" />
  </ExposureAccessLevels>
</AccessProfile>
```

The *level* element refers to a security type defined in the `ClaimSecurityType` typelist or the default type, `unsecuredclaim`.

An `AccessProfile` element contains the following subelements:

Subelement	Description
`ActivityAccessLevels`	Specifies one or more AccessLevel elements. An access level describes a particular relationship with the claim and the access permitted to users who have that relationship.
`ClaimAccessLevels`	Specifies one or more `AccessLevel`, `DraftClaimAccessLevel`, or `ClaimUserAccessLevel`. An access level describes a particular relationship with the claim and the access permitted to users who have that relationship.
`ClaimOwnPermission`	The permission a user must have to own claims of this type. This element is optional.
`ExposureAccessLevels`	Specifies one or more AccessLevel elements. An access level describes a particular relationship with the claim and the access permitted to users who have that relationship.
`SubObjectOwnPermission`	The permission a user must have to own sub objects on this claim. This element is optional.

ClaimOwnPermission and SubObjectOwnPermission

The `ClaimOwnPermission` element restricts claim ownership to users with a specific system permission. For example, sensitive claims might require that only users with `ownsensclaim` **(Permission to sensitive claims)** can own a claim. Similarly, the `SubObjectOwnPermission` element restricts access to exposures and activities to users with a specific system permission. These elements have the following syntax:

```
<AccessProfile securitylevel="level">
  <ClaimOwnPermission permission="perm"/>
  <SubObjectOwnPermission permission="perm"/>
  <ClaimAccessLevels>
...
</AccessProfile>
```

You are not likely to use these elements on a profile for unsecured claims, but they can be useful to control ownership of secured claims. For example, you can restrict ownership of claims involving an employee. Use the following procedure as a model for how to restrict ownership of a claim based on system permissions.

To restrict claim ownership based on system permissions

1. Open a command window to `ClaimCenter\bin`.
2. Execute the following command to launch Studio:
   ```
   gwcc studio
   ```
3. In the Studio **Resources** pane, select **configuration** → **Other Resources** → **security-config.xml**.

4. If you are creating an access profile for a new security type or for a security type that does not yet have an access profile, create an `AccessProfile` element. You can use an existing `AccessProfile` element as a template.

   Otherwise, modify an existing `AccessProfile` element.

5. Create or edit the `ClaimOwnPermission` and `SubObjectOwnPermission` elements, as in the following example:

```
<AccessProfile securitylevel="employeeclaim">
  <ClaimOwnPermission permission="ownemployeeclaim"/>
  <SubObjectOwnPermission permission="ownemployeeclaimsub"/>
  ...
</AccessProfile>
```

   These elements specify which system permission a user must have to own the claim or subobjects of the claim.

6. In ClaimCenter, add the system permissions to the appropriate roles. For example, you might have a role called `Sensitive Claim Adjuster` that has these permissions.

ClaimCenter does not require that you use either the `ClaimOwnPermission` or `SubObjectOwnPermission` element.

AccessLevel Elements for Claims, Exposures, and Activities

The `AccessLevel` subelement can exist within any of the `ClaimAccessLevels`, `ExposureAccessLevels`, and `ActivityAccessLevels` elements. Each of these must contain at least one `AccessLevel` subelement. An `AccessLevel` describes a particular relationship with the claim, exposure, or activity and the access given to users who have that relationship. The format of this element is as follows:

```
<AccessProfile securitylevel="level">
  <ClaimOwnPermission permission="perm"/>
  <SubObjectOwnPermission permission="perm"/>
  <ClaimAccessLevels>
    <AccessLevel level="level" permission="access_type" />
    <DraftClaimAccessLevel level="level"/>
    <ClaimUserAccessLevel role="user_role" level="level" permission="access_type" />
  </ClaimAccessLevels>
  <ActivityAccessLevels>
    <AccessLevel level="level" permission="access_type" />
  </ActivityAccessLevels>
  <ExposureAccessLevels>
     <AccessLevel level="level" permission="access_type" />
  </ExposureAccessLevels>
</AccessProfile>
```

The *level* is the organizational relationship a user must have with the claim, exposure, or activity to access it. This is a value defined in the `ClaimAccessType` typelist. The *level* can be one of the following:

`anyone`	All users.
`group`	Users who belong to the group to which the claim, exposure, or activity is assigned.
`securityzone`	Users who belong to a group in the security zone to which the claim, exposure, or activity is assigned.
`user`	The owner of the claim, exposure, or activity. See “Controlling Document Security” on page 109 for information on additional ownership configuration parameters.

access_type is the claim access type (defined in the `ClaimAccessType` typelist) available to a user with the proper relationship.

ClaimUserAccessLevel

`ClaimAccessLevels` can contain one or more `ClaimUserAccessLevel` subelements. `ClaimUserAccessLevel` elements define the access granted to users listed on the **Users** subtab of the claim. The subtab appears on the

Parties Involved page. One or more `ClaimUserAccessLevel` can appear within the `ClaimAccessLevels` element, and it has the following syntax:

```
<AccessProfile securitylevel="level">
  <ClaimOwnPermission permission="perm"/>
  <SubObjectOwnPermission permission="perm"/>
  <ClaimAccessLevels>
    <AccessLevel level="level" permission="access_type" />
    <DraftClaimAccessLevel level="level"/>
    <ClaimUserAccessLevel role="user_role" level="relationship" permission="access_type" />
  </ClaimAccessLevels>
...
</AccessProfile>
```

The `role` value is the **Roles** code listed for a user on the claim **Users** tab (on **Parties Involved** page). The `UserRole` typelist contains valid values for this attribute. The `level` attribute is the *`relationship`* a user must have with the user in that role. The `level` can be one of the following:

`anyone`	All users.
`group`	Users who belong to the same group as the user with that role.
`securityzone`	Users who belong to the same security zone as the user with that role.
`user`	The user with the specified role.

The `permission` defines the claim access types available to a user with the proper `role` and `level`.

WARNING Be careful adding a `ClaimUserAccessLevel` element. Depending on how you define the element, adding one user through the **Users** tab can grant access to the membership of entire groups or security zones.

DraftClaimAccessLevel

`DraftClaimAccessLevel` defines the access level necessary to work on a claim while its status is draft. This element appears within the `ClaimAccessLevels` element, and has the following syntax:

```
<AccessProfile securitylevel="level">
  <ClaimOwnPermission permission="perm"/>
  <SubObjectOwnPermission permission="perm"/>
  <ClaimAccessLevels>
    <AccessLevel level="level" permission="access_type" />
    <DraftClaimAccessLevel level="level"/>
    <ClaimUserAccessLevel role="user_role" level="level" permission="access_type" />
  </ClaimAccessLevels>
...
</AccessProfile>
```

The *`level`* is the relationship a user must have with the claim creator to access the claim while it is still a draft. These values can be one of the following:

`anyone`	All users.
`group`	All the groups to which the user creating the claim belongs.
`securityzone`	All the security zones to which the user creating the claim belongs.
`user`	The user creating the claim.

Examples of Claim Access Profiles

The following example defines the default `AccessProfile` element for unsecured claims:

```
<AccessProfile securitylevel="unsecuredclaim">
  <ClaimAccessLevels>
    <AccessLevel level="securityzone" permission="view"/>
    <AccessLevel level="securityzone" permission="edit"/>
    <DraftClaimAccessLevel level="securityzone"/>
    <ClaimUserAccessLevel role="subrogationowner" level="user" permission="view"/>
```

```
    <ClaimUserAccessLevel role="subrogationowner" level="user" permission="edit"/>
    <ClaimUserAccessLevel role="reinsmgr" level="group" permission="view"/>
    <ClaimUserAccessLevel role="reinsmgr" level="group" permission="edit"/>
    <ClaimUserAccessLevel role="relateduser" level="user" permission="view"/>
  </ClaimAccessLevels>
  <ActivityAccessLevels>
    <AccessLevel level="group" permission="view"/>
    <AccessLevel level="user" permission="edit"/>
  </ActivityAccessLevels>
  <ExposureAccessLevels>
    <AccessLevel level="group" permission="view"/>
    <AccessLevel level="group" permission="edit"/>
  </ExposureAccessLevels>
</AccessProfile>
```

This example specifies the following for all unsecured claims:

- Any user in the security zone of the claim creator can view and edit the claim.
- Any user in the security zone of the claim creator can access the claim while it is a draft.
- Any user with the subrogation owner role who is added to the claim can view or edit the claim.
- Any member of the same group as a reinsurance manager who is added to the claim can view or edit the claim.
- Any user with the related user role who is added to the claim can view the claim.
- Any user belonging to the same group as an exposure owner can view or edit the exposure.
- Any user belonging to the same group as an activity owner can view the activity, but only the activity owner can edit the activity.

Guidewire provides, by default, the following default access profile for sensitive claims:

```
<AccessProfile securitylevel="sensitiveclaim">
  <ClaimOwnPermission permission="ownsensclaim"/>
  <SubObjectOwnPermission permission="ownsensclaimsub"/>
  <ClaimAccessLevels>
    <AccessLevel level="group" permission="view"/>
    <AccessLevel level="group" permission="edit"/>
    <DraftClaimAccessLevel level="group"/>
  </ClaimAccessLevels>
  <ActivityAccessLevels>
    <AccessLevel level="user" permission="view"/>
    <AccessLevel level="user" permission="edit"/>
  </ActivityAccessLevels>
  <ExposureAccessLevels>
    <AccessLevel level="user" permission="view"/>
    <AccessLevel level="user" permission="edit"/>
  </ExposureAccessLevels>
</AccessProfile>
```

This example specifies the access for all `sensitiveclaim` claims:

- To own the claim or any of its sub objects, a user must have a role that includes the `ownsensclaimsub` system permission.
- Any members of the claim owner's groups can view or edit the claim.
- Only the owner of an exposure can view or edit the exposure.
- Only the owner of an activity can view or edit the activity.

How ClaimCenter Applies Claim ACL Changes

It is important to understand how ClaimCenter applies a change you make to a claim's access control. If you change the access control settings in the `security-config.xml` file, deploy the changes and restart the server. After the restart, new claims are automatically subject to the new ACL. Existing claims remain unchanged. ClaimCenter does not attempt to verify existing claims against the new ACL. ClaimCenter applies new ACL changes to pre-existing claims if a user attempts a claim modification related to the ACL. At that point, ClaimCenter alters the claim's access based on the new configuration but does not flush the old configuration.

You can force ClaimCenter to flush existing access controls and use only the new controls. For example, if you want to apply a new `sensitiveclaim` access control to all claims that pre-date your changes, you would flush the existing access controls.

To flush claim access controls

1. Deploy your new changes and restart the server.
2. Log in to ClaimCenter.
3. Locate an existing sensitive claim.
4. Change the security level of the claim from **Sensitive** to a new value.
5. Save the claim.
6. Set the claim back to **Sensitive**.
7. Save the claim.

You can also flush access controls from an exception rule using the `rebuildClaimACL()` action. See the *ClaimCenter Rules Guide* for information on how to use this action and exception rules.

Controlling Document Security

In addition to the standard document-related system permissions, you can control access to claim documents by configuring document access permissions. To use this feature, a document must have a document security type set. To see a set of documents of a particular type, a user must have *both* permission to view documents generally *and* access to the document security type.

A document type is set using the document's **Security Type** on the user interface or through a Gosu class that you write. A `DocumentPermission` element in the `security-config.xml` file controls access to a document type. This element has the following syntax:

```
<DocumentPermissions>
  <DocumentAccessProfile securitylevel="level">
    <DocumentViewPermission permission="perm"/>
    <DocumentEditPermission permission="perm"/>
    <DocumentDeletePermission permission="perm"/>
  </DocumentAccessProfile>
</DocumentPermissions>
```

level specifies a document security type defined in `ClaimCenter/modules/cc/config/typelists/DocumentSecurityType.ttx`. You can also refer to the typelists section of the *ClaimCenter Data Dictionary*. You must have one `DocumentAccessProfile` for each document security type in the typelist.

The `DocumentAccessProfile` element can contain from zero to three security parameters:

- `DocumentViewPermission`
- `DocumentEditPermission`
- `DocumentDeletePermission`

These each have the format `<DocumentViewPermission permission="perm">`, in which *perm* is the permission required for the view, edit, or delete operation on the document with the specified level. Guidewire provides the following the default configuration for you:

```
<DocumentPermissions>
  <DocumentAccessProfile securitylevel="unrestricted"/>
  <DocumentAccessProfile securitylevel="internalonly">
    <DocumentViewPermission permission="viewsensdoc"/>
    <DocumentEditPermission permission="editsensdoc"/>
    <DocumentDeletePermission permission="delsensdoc"/>
  </DocumentAccessProfile>
  <DocumentAccessProfile securitylevel="sensitive">
    <DocumentViewPermission permission="viewsensdoc"/>
    <DocumentEditPermission permission="editsensdoc"/>
    <DocumentDeletePermission permission="delsensdoc"/>
```

```
    </DocumentAccessProfile>
  </DocumentPermissions>
```

In the default, unrestricted documents have no access restrictions. Documents marked `sensitive` require that the user have at least the `viewsensdoc` permission to view, `editsensdoc` to edit, and `delsensdoc` to delete. Documents at the `unrestricted` level require no special access permissions at all.

Note: ClaimCenter grants these permissions based on the user's roles alone. You cannot restrict document access based on security zones or groups.

Example of Controlling Document Security

Suppose you have three groups that access claims and attach documents to them: **Adjusters**, **Subrogation**, and **Special Investigations**. The subrogation and special investigation documents are confidential and you want them to be viewed only by members of their respective groups. So, for example, if you have a single claim with 6 documents:

- 3 documents added by the **Adjusters**
- 1 document added by **Special Investigations**
- 2 document added **Subrogation**

You want to configure security such that a **Subrogation** user sees 5 (3+2) documents, **Special Investigations** 4 (1+3), and **Adjusters** see only 3. To take the example further, you want a member of the **Managers** group to see all 6 documents. The process for creating such a configuration includes:

- Create supporting document security types and system permissions.
- Create a `DocumentAccessProfile` that restricts access to the new document types.
- Regenerate the toolkit, rebuild and redeploy ClaimCenter.
- Add the new permissions to the appropriate roles.

The following procedures illustrate how you can implement this example.

To create document security types

For this example, use the procedure to create subrogation and special investigations document security types with the following values:

Code	Name	Description
`subrogation`	`Subrogation Doc`	`subrogation document`
`specialinv`	`Special Inv Doc`	`special investigations document`

1. Open a command window to `ClaimCenter\bin`.
2. Execute the following command to launch Studio:
   ```
   gwcc studio
   ```
3. In the Studio **Resources** pane, select **configuration** → **Typelists** → **DocumentSecurityType**.
4. On the **Codes** tab of the **DocumentSecurityType** pane, click **Add**.
5. Enter a **Code** for the new access type. The **Code** must be 16 characters or less.
6. Add a **Name** and **Description** to the new security type. Leave the **Priority** as `-1` and **Retired** as `false`.
7. Select **File** → **Save Changes**.
8. Rebuild and redeploy your application package file. If you are in a test environment, you can just copy your configuration. In both cases, run `regen-dictionary` to update the typelist documentation.

To create custom permissions for the new document security types

For this example, use the procedure to create custom permission types with the following values:

Code	Name	Description
viewsubdoc	View subrogation documents	Permission to view a subrogation document
editsubdoc	Edit subrogation documents	Permission to edit a subrogation document
delsubdoc	Delete subrogation documents	Permission to delete a subrogation document
viewspecinvdoc	View special inv documents	Permission to view a special inv document
editspecinvdoc	Edit special inv documents	Permission to edit a special inv document
delspecinvdoc	Delete special inv documents	Permission to delete a special inv document

1. Open a command window to `ClaimCenter\bin`.
2. Execute the following command to launch Studio:
   ```
   gwcc studio
   ```
3. In the Studio **Resources** pane, select **configuration** → **Typelists** → **SystemPermissionType**.
4. On the **Codes** tab of the **DocumentSecurityType** pane, click **Add**.
5. Enter a **Code** for the new access type. The **Code** must be 16 characters or less.
6. Add a **Name** and **Description** to the new security type. Leave the **Priority** as `-1` and **Retired** as `false`.
7. Select **File** → **Save Changes**.
8. Rebuild and redeploy your application package file. If you are in a test environment, you can just copy your configuration. In both cases, run `regen-dictionary` to update the typelist documentation.

To create a DocumentAccessProfile for the new document types

1. Open a command window to `ClaimCenter\bin`.
2. Execute the following command to launch Studio:
   ```
   gwcc studio
   ```
3. In the Studio **Resources** pane, select **configuration** → **Other Resources** → **security-config.xml**.
4. Add a `subrogation` `DocumentAccessProfile` element to the `DocumentPermissions`.
5. Add a `specialinv` `DocumentAccessProfile`.

   The finished `DocumentPermissions` block looks like the following:

```
<DocumentPermissions>
  <DocumentAccessProfile securitylevel="unrestricted"/>
   ...
  <DocumentAccessProfile securitylevel="subrogation">
     <DocumentViewPermission permission="viewsubdoc" />
     <DocumentEditPermission permission="editsubdoc"/>
     <DocumentDeletePermission permission="delsubdoc"/>
  </DocumentAccessProfile>
  <DocumentAccessProfile securitylevel="specialinv">
     <DocumentViewPermission permission="viewspecinvdoc" />
     <DocumentEditPermission permission="editspecinvdoc"/>
     <DocumentDeletePermission permission="delspecinvdoc"/>
  </DocumentAccessProfile>
</DocumentPermissions>
```

6. Select **File** → **Save Changes**.
7. Rebuild and redeploy your application package file. If you are in a test environment, you can just copy your configuration. In both cases, run `regen-dictionary` to update the documentation.

To add the new permissions to the appropriate roles

1. If you have not already done so, start ClaimCenter.
2. Add the new permissions to the appropriate roles. For example, add the subrogation system permissions to the **Subrogator** role, or whichever role is appropriate for your configuration.
3. Repeat step 2 for the special investigation permissions adding them to the appropriate role.
4. Add the subrogation and special investigation permissions to the **Manager** role.
5. Ensure that the members of each group have the associated roles.

At this point, you can test the new configuration.

Controlling Exposure Security

Your company might want to refine the controls that allow users to access exposures on a claim. For example, you might want to restrict adjusters working on different exposures on the same claim to only exposures of a certain type. You can do this through exposure access control.

What Exposure Security Controls

Exposure level access control restricts access to exposures within a claim. This means an adjuster on a claim can have access to some exposures on a claim but not all. Users granted access through exposure access control can view:

- the exposure screen
- the existence and contents of all notes related to a secured exposure
- the existence and contents of all documents tied to a secured exposure
- the contents of activities related to a secured exposure
- the contents of matters related to an exposure

Exposure access control does not prevent users from viewing any of the following:

- the existence of exposures that they are not allowed to see (a claim lists all exposures)
- financial transactions related to an exposure that they are not allowed to see

If users attempt to access an exposure to which they do not have access, then they receive a permissions error.

Note: Guidewire added two new system permissions to control access to exposures: `expview` and `expedit`.

How to Use Exposure Security

To use exposure access control:

- Define one or more exposure security types in the `ExposureSecurityType` typelist.
- Provide a mechanism for setting the security type on an exposure. You can do this with a Gosu rule or by exposing the field directly on the user interface and allowing the user to set it. You can also use a combination of both techniques.
- Implement static or claim-based exposure security.

Regardless of whether you implement static or claim-based exposure security, you must create an ExposurePermission element in the security-config.xml file. The ExposurePermission must come last in the security-config.xml file and has the following format:

```
<ExposurePermissions>
  <ExposurePermission securitylevel="type" permission="perm"/>
</ExposurePermissions>
```

The *level* parameter is optional and represents an exposure security type. The *perm* parameter refers to an optional system permission required to access exposures at the specified *level*. The following example illustrates a typical set up:

```
<ExposurePermissions>
  <ExposurePermission securitylevel="secured" permission="expeditsec"/>
  <ExposurePermission permission="unsecexpedit"/>
</ExposurePermissions>
```

To access a secured exposure (or its related objects), users also need to have the expeditsec permission on at least one role.

ClaimCenter considers exposures with an undefined securitylevel as having the null security level. To assign a permission to these exposures, specify an ExposurePermission element without a securitylevel parameter. Only define one ExposurePermission that omits the securitylevel. If you specify more than one, ClaimCenter assigns the last one encountered to exposures at the null level.

Note: You can only map each security type one time. This means you can map secured exposures to expeditsec but not to viewsecexp.

Static Versus Claim-based Exposure Security

You can implement static or claim-based exposure security. A static implementation is a simpler method and only requires you to construct an ExposurePermissions element. Using the previous example, suppose an exposure has a securitylevel of secured. ClaimCenter allows a user to access an exposure if the user has the expeditsec permission.

To implement claim-based exposure security, you take the mapping a step further. In this method, you map the exposure security type to a system permission which you then map to a claim access type. For example, you can implement the claim-based method using the abexposure exposure security type provided by Guidewire.

To implement claim-based exposure security

1. Add a custom system permission called abexposures in the SystemPermissionType typelist.
2. Create an ExposurePermissions element in the security-config.xml file:

```
<ExposurePermissions>
  <ExposurePermission securitylevel="abexposure" permission="abexposures"/>
</ExposurePermissions>
```

3. Create an abexposure type code in the ClaimAccessType typelist.

```
<typecode code="abexposure" name="Test" desc="Custom permission"/>
```

ClaimCenter allows you to create a claim access type and an exposure security type with the same typecode.

4. Create an AccessMapping element in the security-config.xml file:

```
<AccessMapping claimAccessType="abexposure" systemPermission="abexposures"/>
```

5. An AccessProfile element that references this access type:

```
<AccessProfile securitylevel="sensitiveclaim">
  ...
  <ExposureAccessLevels>
    <AccessLevel level="user" permission="abexposure"/>
  </ExposureAccessLevels>
</AccessProfile>
```

If this access control is in place, then users must have both the abexposures permission and access to the particular claim to access an exposure. The abexposures permission grants access to exposures. The ClaimAccessProfile element defines access to the particular claim.

Guidewire recommends that you implement the claim-based method only with custom claim access types. This mean you would need one custom claim access type for each exposure security type. Many custom claim access types can put a performance load on the system. Use this method with care.

Note: You cannot simply use the default claim access types for claim-based exposure security. Mapping the default claim access type `view` to both the `abexposures` and the `expview` would eliminate the distinction between all claim exposures and `abexposure` exposures.

chapter 10

Importing and Exporting Administrative Data

This topic discusses the key concepts and procedures you need to know about importing administrative data from an external system into ClaimCenter. While users enter much of the information into ClaimCenter directly, at times it is more convenient or necessary to enter information in bulk. For example, if you receive new claim information from an external system, such as a call center FNOL system, you can import these new claims into ClaimCenter in bulk. This topic also discusses how to export data.

This topic includes:

Understanding Data Import and Export

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

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

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

You must edit and load these files while first setting up ClaimCenter. Later, you might want to change this information. For example, you might change the permissions granted to the `Adjuster` role or add a new activity pattern. In this case, you can edit the file and reimport it instead of manually making changes. The import capability is particularly useful for a development team preparing a new ClaimCenter installation or upgrading an existing one.

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

What Mechanisms are Available to Import and Export Data?

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

Method	Description
CSV files	You create one or more CSV files and upload them using the `import_tools` command. This method is useful for development environments in which multiple people share sample data. For a detailed explanation see “Importing Administrative Data from the Command Line” on page 118.
user interface	You create one or more import files containing administrative data, and you import them using the **Import/Export Data** functionality provided on the **Administration** tab. You must have the `viewadmin` and `soapadmin` permissions to access this option. This method also warns against and enables you to resolve collisions between incoming data and existing data. You can also use the user interface to export administrative data. For a detailed explanation, see “Importing and Exporting Administrative Data from ClaimCenter” on page 121. You can also import sample data for use in development. See “Installing Sample Data” on page 39 in the *Installation Guide*.
staging tables (zone data only)	You can use the functionality provided by the `ITableImportAPI` interface to import administrative data. See “*Importing from Database Staging Tables*” in the *ClaimCenter Integration Guide* for detailed information about using this mechanism. This API is only for use to import administrative data. Importing other types of data using this API is not supported. You can also import administrative data from the staging tables using the table_import command. See “table_import Command” on page 176.
Gosu	You can create one or more Gosu entity builder classes that define data. See “Using Entity Builders to Create Test Data” on page 568 in the *Configuration Guide*for a detailed explanation of this method.

The ClaimCenter Data Model

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

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

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

```
gwcc regen-dictionary
```

This reference describes the structure of business objects stored by ClaimCenter and defines the properties and foreign key references for each object. See the *ClaimCenter Configuration Guide* for more information about using the *ClaimCenter Data Dictionary.*

If you change the data model, regenerate the *ClaimCenter Data Dictionary.*

Public ID Prefix

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

For example, an external system might have a vehicle (`V1`) associated with a specific policy (`P1`). The import data must be structured such that ClaimCenter knows that vehicle `V1` is associated with policy `P1`. Additionally, the external system might need to locate or even update objects in a ClaimCenter database after the initial import. For example, if a vehicle `V2` is added to the policy `P1`.

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

origin:*ID*

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

> **Note:** The IDs need to be unique only within objects of the same type. For example, all policies must have a different public ID. However, a claimand 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 ClaimCenter configuration, multiple developers can create administrative data in their own ClaimCenter installations. Administrative data includes users, groups, roles and so forth. You combine this data into a single production database at the end of the development cycle.

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

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

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

{*PublicIDPrefix*}:{*ID*}

Using the public ID prefix correctly requires some coordination among engineers to ensure that no prefixes overlap. For example, engineers might use their initials or computer names as a public id prefix.

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

What Happens During an Import?

As ClaimCenter loads external data from the command line, it uses the public ID to determine whether it already knows about the object. If ClaimCenter finds a match, it reinserts the existing record instead of adding a new

record. When ClaimCenter reinserts the data, it overwrites any existing values. If there are differences, the incoming values overwrite the current values in the ClaimCenter database.

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

You can use the **Import/Export Data** option on the **ClaimCenter Administration** page instead of the command line commands to import administrative data. You must have the `viewadmin` and `soapadmin` permissions to access this feature. Using this page, ClaimCenter enables you to step through differences between imported data and data in the database. At this point, you can choose to accept the imported data or keep what is in the database.

Importing Administrative Data from the Command Line

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

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

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

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

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

Creating a CSV File for Import

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

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

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

- without an `entityid` string
- containing comma delimited values
- containing a value in its third comma-delimited field

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

Constructing a Heading

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

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

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

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

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

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

Working with Data for Import

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

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

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

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

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

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

publicID

or

entity_id:identity_source

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

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

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

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

Simple Array Data. You specify simple array data, referencing a single foreign key, using the following format:

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

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

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

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

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

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

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

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

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

Using CSV Files with Different Character Sets

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

Maintaining Data Integrity While Importing

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

To maintain data integrity while importing data

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

Importing and Exporting Administrative Data from ClaimCenter

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

Creating an XML File for Import

The `ClaimCenter/build/xsd/cc_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: `cc_entities.xsd` and `cc_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.

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. ClaimCenter reports errors only if a referenced object still does not exist after reading the entire file.

Validating the XML

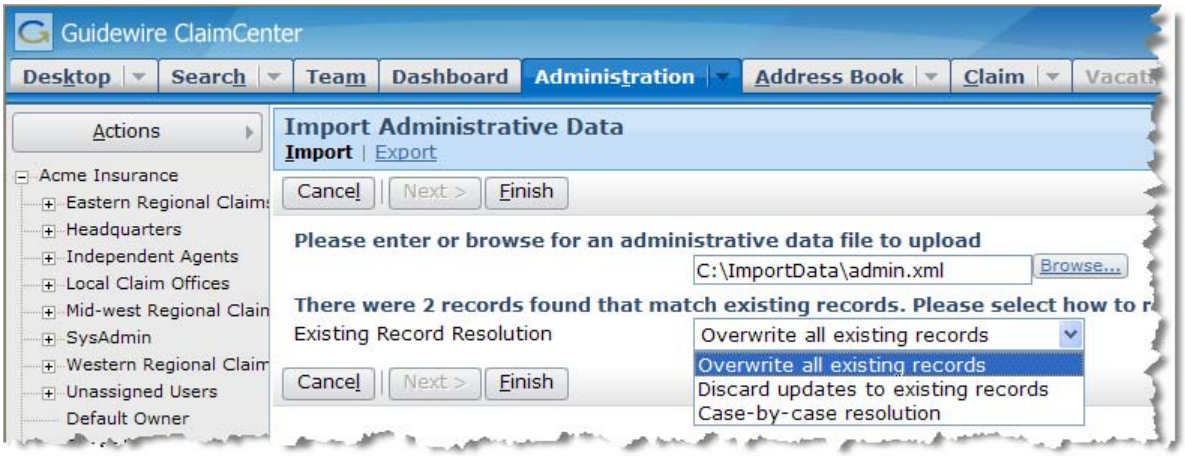
You can validate the XML of your import file against an `cc_import.xsd` file using the following code:

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

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

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

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

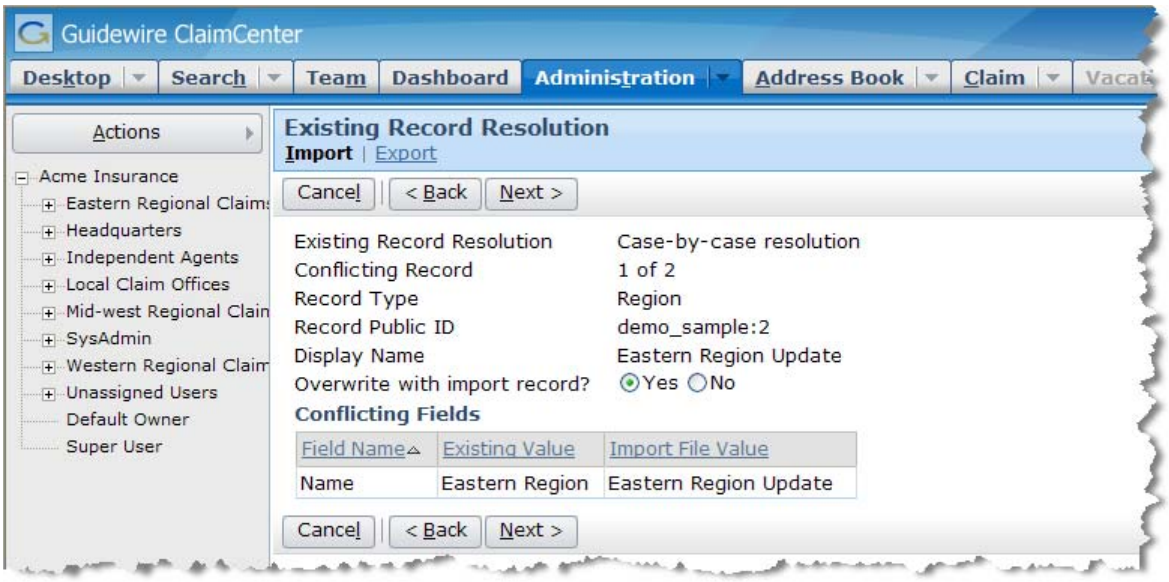
Importing Data From the User Interface

You can import administrative data from the user interface. Log on as a user with the `viewadmin` and `soapadmin` permissions and choose **Import/Export Data** from the **Administration** page.

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



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



During import, ClaimCenter inserts a new entity or updates an existing entity for each entity in the XML file, just as with the command-line import. During an import, ClaimCenter does not run validation rules. However ClaimCenter does run pre-update rules. For this reason, run the `maintenance_tools` actions to check for user and group exceptions after you import administrative data.

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

Exporting Data from the User Interface

You can export different sets of data, including:

- **Admin** - includes all of the other categories
- **Authority Limit Profiles**
- **Business Weeks**
- **Catastrophes**
- **Coverage Verifications**
- **Exchange Rates**
- **Holidays**
- **ICD 9 Codes**
- **Large Loss Thresholds**
- **Metric Limits**
- **Questions**
- **Regions**
- **Reinsurance Thresholds**
- **Roles**
- **Users and Groups**

IMPORTANT If a particular data set is not on the previous list, you cannot export it using this function.

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

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

To export administrative data from ClaimCenter

1. Log on as a user with the `viewadmin` and `soapadmin` permissions
2. Click the **Administration** tab.
3. Choose **Import/Export Data**.
4. Select the **Export** tab.
5. Select the data set to export.
6. Click **Export**. Your browser prompts you to open or download the file.
7. Select to download the XML file.

Other Import Functions

This section introduces other import functions available with ClaimCenter.

Importing a Table from an External Database

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

- check the integrity of data in the staging tables
- populate an exclusion table
- delete rows from the staging table based on an exclusion table
- run an integrity check and then load the data into ClaimCenter

Guidewire provides the `ITableImportAPI` that performs all of the same functions as this command. This API enables you to develop integrations that can do regular imports from external databases. See the *ClaimCenter Integration Guide* for detailed information about this API and converting data from external sources.

Note: This command requires that you put the server in maintenance mode using the `-maintenance` flag with the `system_tools` command.

The import Directory

When you install ClaimCenter, the database upgrader loads default roles, security zones, and privileges. After the install, you can use the **Import/Export Data** option on the **Administration** tab operations to change these values. You must have the `viewadmin` and `soapadmin` permissions to access this feature. You can modify files to bypass the user interface if you need to make large scale changes. Unless otherwise noted, these files can be accessed from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources** → **import**. Make the changes in the files from Studio and reimport the files with the `import_tools` command as follows:

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

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

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

WARNING Do not edit any of the files within the `ClaimCenter/modules/cc/config` directory as this corrupts the base installation. Only modify these files using Guidewire Studio. Studio copies the file to `ClaimCenter/modules/configuration/config` any time you edit the file. This creates an implementation-specific version of the file, leaving the base version intact.

Configuring Roles and Privileges

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

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

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

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

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

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

Managing Authority Limit Profiles

You manage authority limit profiles in the `authority-limits.csv file.` This file has the same format as the sample data files. Profile definitions consist of 2 sets of data: an a `AuthorityLimit` profile row and `AuthorityLimit` rows:

```
Authority Limit Profiles,,,,,,,,,
type,data-set,entityid,name,description,,,,,
AuthorityLimitProfile,0,default_data:1,Adjuster,Adjuster default authority,,,,,
AuthorityLimitProfile,0,default_data:2,Claims Supervisor,Claims supervisor default authority,,,,,
AuthorityLimitProfile,0,default_data:3,Regional Supervisor,Regional supervisor default authority,,,,,

type,data-set,entityid,user,profile,claim,limittype,coveragetype,costtype,limitamount
AuthorityLimit,0,default_data:1,,default_data:1,,ctr,,,15000
AuthorityLimit,0,default_data:2,,default_data:1,,cptd,,,15000
AuthorityLimit,0,default_data:4,,default_data:2,,ctr,,,25000
AuthorityLimit,0,default_data:5,,default_data:2,,cptd,,,25000
AuthorityLimit,0,default_data:7,,default_data:3,,ctr,,,100000
AuthorityLimit,0,default_data:8,,default_data:3,,cptd,,,100000
```

The `name` and `description` are solely for your benefit in understanding which types of users typically get these limits. In the previous sample, the `adjuster` profile is being given a $15,000 limit for total claim total reserves (`ctr`) for general liability coverage. See the *ClaimCenter Data Dictionary* for the `AuthorityLimitType` typelist

for the possible limit type codes (for example, `ctr`). The `AuthorityLimitType` typelist contains the available codes. See the *ClaimCenter Data Dictionary* for details. The `costtype` and `coveragetype` fields similarly contain code values from these typelists. You need not specify a `costtype` or `coveragetype` you can, as these samples illustrate, simply leave these value empty.

Configuring Security Zones

You can use the `security-zones.xml` file to manage security zones. Security zones enforce data security by categorizing groups into different organizational sections. A complete security zone definition appears as follows:

```
<SecurityZone public-id="default_data:1">
  <description>Office A</description>
  <name>Security Zone for Office A</name>
</SecurityZone>
```

Again, you use the *`name`* and *`description`* only to remind you in which security zone to place a group.

Access `security-zones.xml` from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**.

Updating ICD Codes

ClaimCenter includes ICD-9-CM (International Classification of Diseases, Ninth Revision, Clinical Modification) codes (Volumes 1 and 2) updated for use as of October 1, 2009, for federal fiscal year 2010. Annually, the CDC releases a revised set of codes, available from their website:

`http://www.cdc.gov/nchs/icd/icd9cm.htm`

The revised set includes invalid, updated, and new codes. This topic describes how to update ClaimCenter with these changes.

The Centers for Medicare and Medicaid Services (CMS) agency of the U.S. Department of Health and Human Services (HHS) provides lists of the invalid, updated, and new codes in separate PDF files from their website:

`http://www.cms.hhs.gov/icd9ProviderDiagnosticCodes/07_summarytables.asp`

Note: You will need Adobe Acrobat Professional or a similar application to save the PDF files in a different format.

Importing New ICD Codes

To import new codes

1. Open the PDF file of new codes.
2. If using Adobe Acrobat Professional, export the file to RTF (Rich Text Format). Alternatively, you can copy the data into an RTF, page by page.
3. You may want to convert all the text to uppercase so that the data looks consistent with older codes. However, this step is not required.
4. Copy the tables from the RTF into Microsoft Excel.
5. Set the code column to text data type, otherwise all leading and trailing zeros are truncated.
6. The column format needs to be as follows to load correctly into the `ICDCode` table:

   `type,data-set,entityid,bodysystem,code,codedesc,chronic,expirydate,availabilitydate`
7. Use the following values:

Key	Value
`type`	`ICDCode`

Key	Value
data-set	0
entityid	icd:*xxxx* This is the unique ID for the table. Check the last row in the existing ICD CSV in ClaimCenter\cc\config\referencedata and increment by 1 for each row. You can use the Microsoft Excel auto-increment functionality to create entity IDs beyond the first one.
bodysystem	Add the numeric value that corresponds to the range the ICD code falls into. Review the ICDBodySystem typelist for code ranges.
code	This is from the CMS website.
codedesc	This is from the CMS website.
chronic	Leave blank.
expirydate	Leave blank.
availabilitydate	Add the date when the code will be available, such as 10/1/2010. Replicate for all rows of data.

8. Save this worksheet as a CSV file in ClaimCenter\configuration\config\referencedata.
9. Open a command window to ClaimCenter\admin\bin.
10. Run the following command to import the CSV file.

```
import_tools -password password -import ClaimCenter\configuration\config\referencedata\fileName
```

Updating ICD Code Descriptions

To update code descriptions

1. Open the PDF file of revised diagnosis code titles. The title is stored in ClaimCenter as the description.
2. Log in to ClaimCenter as a user with **View Admin**, **View reference data**, and **Edit reference data** permissions.
3. Click the **Administration** tab.
4. Click **ICD Codes**.
5. For each code listed in the PDF, enter the code in the **Code** field.
6. Click **Search**.
7. Click the **ICD Code** value.
8. Click **Edit**.
9. Copy the description from the PDF and paste it into the **Description** field in ClaimCenter.
10. Click **Update**.
11. Repeat steps 5 through 11 until you have reached the end of the PDF.

Expiring Invalid ICD Codes

To delete invalid ICD codes

1. Open the PDF file of invalid diagnosis codes.
2. Log in to ClaimCenter as a user with **View Admin**, **View reference data**, and **Edit reference data** permissions.

3. Click the **Administration** tab.
4. Click **ICD Codes**.
5. For each code listed in the PDF, enter the code in the **Code** field.
6. Click **Search**.
7. Click the **ICD Code** value.
8. Click **Edit**.
9. Set the **Expires On** date.
10. Click **Update**.
11. Repeat steps 5 through 10 until you have reached the end of the PDF.

chapter 11

Batch Processes and Work Queues

This topic explains the ClaimCenter batch processing tools. There are several ClaimCenter processes that you can schedule to run in the background on a regular basis. For example, you might want to calculate the latest statistics every hour, or perform a nightly check for any claim exceptions that need to be raised.

This topic includes:

Understanding Batch Processes

A batch process is a background process that performs important tasks independent of an user or administrator. All batch processes execute on the batch server. A batch process runs to completion and then reports its result back to a log or to the user interface.

ClaimCenter uses batch processing to perform many repetitive tasks that can impact large amounts of data. For example, a batch process could check for expirations on object due dates and flag expired objects for further work.

You can schedule many batch processes by using the ClaimCenter scheduler. The scheduler starts batch processes according to the schedule defined in `scheduler-config.xml`. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources.** See “Scheduling Batch Processes and Distributed Work Queues” on page 143. If it is necessary to process items outside of the normal schedule, you can use the command-line or user interface to start a batch process. See “maintenance_tools Command” on page 171 for instructions on how to launch a batch process from the command line. See “Batch Process Info” on page 152 for instructions on how to launch batch processes from ClaimCenter.

ClaimCenter comes with a number of batch processes and distributed work queues. See “Batch Processes and Distributed Work Queues” on page 134 for descriptions.

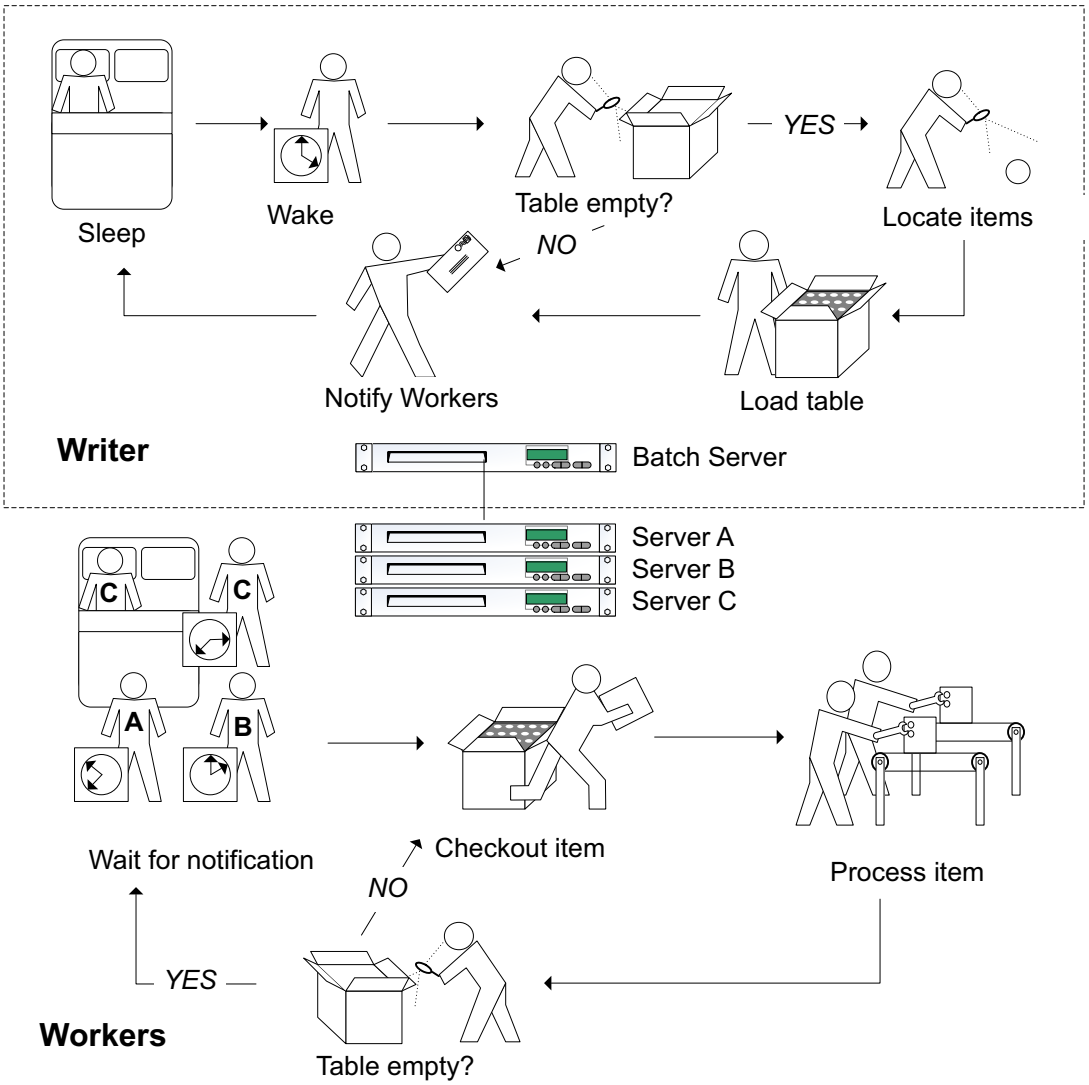
You can also create custom batch processes. See “Custom Batch Processes” on page 325 in the *Integration Guide*.

Understanding Distributed Work Queues

Some batch processes do all their work on the batch server within a single process. For processing-intensive tasks, a work queue distributes processing across multiple servers. Guidewire calls this a *distributed work queue*.

In a ClaimCenter cluster, worker processes can run in parallel on multiple servers. Worker processes are not restricted to running on the batch server. The batch process that supports workers is a *writer.* A writer is a schedulable batch process that runs only on the batch server. A writer loads a queue with work items to be completed by one or more workers. Together, the arrangement of workers, a writer, and a queue make up a distributed work queue. Distributed work queues have an advantage because multiple servers in a cluster can share processing. Each work queue concerns itself with a specific task.

The following diagram illustrates how a distributed work queue functions:



When the work queue's writer wakes, it first checks the `WorkItem` table to see if it contains any items specific to its task. If the table is not empty, the writer notifies the workers and goes back to sleep. If the table does not contain any work items, the writer:

1. Locates items for processing.
2. Loads the items into the `WorkItem` table.
3. Notifies the workers.
4. Goes back to sleep.

Workers only process items associated with their writer. So, for example, the ActivityEsc writer might have six workers on three different servers in a ClaimCenter cluster. Those workers work only on ActivityEsc items. When a worker wakes, it first checks the `WorkItem` table in the ClaimCenter database to see if it contains any items from its associated writer. If it does not, the worker goes back to sleep. Typically, there are multiple worker processes for every writer such that there are usually one or more workers processing at some point throughout the day.

If a worker wakes and there are items available for processing, the worker checks out its allotted item quota. 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 at checkout.
`CheckedOutBy`	Set to the worker.

After checking out its item allotment, the worker then begins processing them individually. When a worker completes processing an item, it removes it from the table and begins to process the next item.

A distributed queue's writer wakes up at the interval specified in `scheduler-config.xml`. Typically, you schedule the writer to wake up one or two times a day. The writer process runs on the batch server, just as any other batch process. For more information, see "Scheduling Batch Processes and Distributed Work Queues" on page 143.

Worker processes wake up much more frequently, at least every `maxpollinterval`. After a worker wakes it can process up to `batchsize` items. You configure these and other values for the distributed work queue in the `work-queue.xml` file. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**. See "Configuring Distributed Work Queues" on page 133 for detailed information about configuring a work queue.

Running Batch Processes and Work Queues

You can launch many batch processes, including writer processes for distributed work queues, from ClaimCenter or from the command line.

Running Batch Processes and Work Queues from ClaimCenter

You can run many batch processes, including writer processes for distributed work queues, from the **Batch Process Info** screen in ClaimCenter.

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 have access to the **Server Tools** pages. For more information about server modes, see "Understanding Server Run Levels and Modes" on page 62.

To run a batch process from the Batch Process Info page

1. Log in to ClaimCenter.
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`.

The **Batch Process Info** screen contains information such as the current status of the batch process, when it last ran, when it will run again, and the schedule. See "Batch Process Info" on page 152.

Running Batch Processes and Work Queues from the Command Line

Use the `maintenance_tools` command to run a batch process, including writer processes for distributed work queues, from the command line.

To run a batch process from the command line

1. Start the ClaimCenter server if it is not already running.
2. Open a command window.
3. Navigate to `ClaimCenter/admin/bin`.
4. Run the following command:

```
maintenance_tools -password password -startprocess process
```

For the *process* value, specify a valid process code. For a list of batch process codes, including work queue writer processes, see "Batch Processes and Distributed Work Queues" on page 134.

Terminating Batch Processes and Work Queues from the Command Line

You can terminate many in-progress batch processes using the `maintenance_tools` command.

To terminate a batch process from the command line

1. Open a command window.
2. Navigate to `ClaimCenter/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. For a list of batch process codes, including work queue writer processes, see "Batch Processes and Distributed Work Queues" on page 134.

Single phase processes cannot be terminated using this command. Single phase processes run in a single transaction, so there is no convenient place to terminate the process. The following batch processes are single phase processes that cannot be terminated:

- `AggregateLimitCalculation`
- `DashboardStats`
- `DataDistribution`
- `ExchangeRate`
- `FinancialsCalculation`
- `Statistics`
- `Table Import`

Configuring Distributed Work Queues

To configure a distributed work queue, you configure a schedule for the writer batch process, the `progressinterval` of the work queue, and the worker processes. Configure the writer batch process schedule in the `scheduler-config.xml` file, just as you would any other batch process. See "Scheduling Batch Processes and Distributed Work Queues" on page 143. Configure worker processes in the `work-queue.xml` file. Both files can be accessed from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**. After you edit either file, you must rebuild and redeploy ClaimCenter.

Each distributed 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 takes a required `workQueueClass` and `progressinterval` value.

The workQueueClass must be one of the Guidewire-provided work queue classes listed in work-queue.xml. These work queue classes implement the BatchProcess interface. You cannot configure custom batch processes as distributed work queues. You also cannot configure Guidewire-provided batch processes as distributed work queues if they are not already defined as distributed work queues in work-queue.xml.

The progressinterval value is the amount of time, in milliseconds, that ClaimCenter allots for a worker to process batchsize work items. If the time a worker has held a batch of items exceeds the progressinterval, then ClaimCenter considers the work items orphans. ClaimCenter reassigns the orphan work items to a new worker instance. Therefore, set the progressinterval larger than the longest time required to process a work item multiplied by the batchsize.

The workItemRetryLimit parameter in config.xml defines a limit on the number of times that work queues retry processing an orphaned work item.

For each work queue, you can declare as many worker instances as you want, specifying on which server and environment each one runs. If you do not specify a worker for a queue, or you do not specify the env and server attributes, ClaimCenter starts a single instance on the batch server. See “Using the registry Element to Specify Environment Properties” on page 18 for information about the env and server attributes. The worker subelements take the following attributes, all of which are optional:

Attribute	Description
instances	The number of workers to create. By default, ClaimCenter always creates at least one worker.
minpollinterval	How long, in milliseconds, a worker must sleep after exhausting all work items, ignoring notifications, before waking up and polling the database again for more work items. For a system generating many work items and notifications and configured with many workers, increase this attribute from the default value of 0. This reduces unnecessary load on the database. The expected latency to process a work item is minpollinterval / instances.
maxpollinterval	How often workers wake up automatically, even if the workers receive no notification. This default might not be appropriate for large numbers of workers. You might need to increase maxpollinterval to prevent excessive numbers of queries for work items. The default maxpollinterval is 60,000 milliseconds.
throttleinterval	The delay between processing workitems in milliseconds. The value controls how long the process sleeps. A value of 0 (zero) means worker processes work items as rapidly as possible. To reduce the CPU load, set the throttleinterval to a positive value.
batchsize	How many work items the worker attempts to check out while searching for more work items. Larger batch sizes are more efficient, but might not result in good load distribution. The default batchsize is 10.
env	The environment in which this particular worker is active.
server	The server on which this particular worker is active.

You can dynamically configure work queues by using web services. See “Manipulating Work Queues Using Web Services” on page 84 in the *Integration Guide*.

Batch Processes and Distributed Work Queues

ClaimCenter has a number of batch processes and work queues that you can run from the command line or at a scheduled time. For information on financial transaction statuses, see “Lifecycles of Financial Objects” on page 156 in the *Application Guide*.

To start, stop, or get the status of a batch process or work queue, use the maintenance_tools command. See “maintenance_tools Command” on page 171. You can also call the IMaintenanceToolsAPI directly. See “Starting or Querying Batch Processes Using Web Services” on page 84 in the *Integration Guide*.

The following sections describe the batch processes and distributed work queues provided with ClaimCenter. Many of these batch processes and work queues can be launched from the **Batch Process Info** or **Work Queue Info** page.

These pages are available from the **Server Tools**. Users with the `internaltools` permission can access the **Server Tools** by pressing `ALT+SHIFT+T`. The **Admin** role has this permission by default. When you press this key combination, ClaimCenter displays the **Server Tools** tab. Click **Batch Process Info** to access the **Batch Process Info** page. Or, click **Work Queue Info** to access the **Work Queue Info** page.

See also

- “Batch Process Info” on page 152
- “Work Queue Info” on page 152

Activity Escalation

Code: `ActivityEsc`

This batch process is the writer for the Activity Escalation distributed work queue. The Activity Escalation work queue finds activities that are open, not yet escalated, and past the escalation time and runs activity escalation rules on matching activities. By default, this process runs every 30 minutes.

Aggregate Limit Calculations

Code: `AggLimitCalc`

Forces a recalculation of all limits used and repopulates the database tables. Run the Aggregate Limit Calculations process only if you encounter consistency check failures and can not identify the reason for the inconsistency.

The server must be at the maintenance run level to run Aggregate Limit Calculations. You cannot run this process from the ClaimCenter user interface or schedule the process. You can only run Aggregate Limit Calculations from the command line or by calling `IMaintenanceToolsAPI`.

See also

- “Using the Maintenance Run Level” on page 64
- “Running Batch Processes and Work Queues from the Command Line” on page 132
- “Starting or Querying Batch Processes Using Web Services” on page 84 in the *Integration Guide*

Aggregate Limit Loader Calculations

Code: `AggLimitLoaderCalc`

Rebuilds the aggregate limit calculations. Always run this process after you import financial and limit information.

The server must be at the maintenance run level to run Aggregate Limit Loader Calculations. You cannot run this process from the ClaimCenter user interface or schedule the process. You can only run Aggregate Limit Loader Calculations from the command line or by calling `IMaintenanceToolsAPI`.

See also

- “Using the Maintenance Run Level” on page 64
- “Running Batch Processes and Work Queues from the Command Line” on page 132
- “Starting or Querying Batch Processes Using Web Services” on page 84 in the *Integration Guide*

Archiving Item Writer

Code: `Archive`

This batch process is the writer for the Archive distributed work queue. The archive work queue writes closed claims and associated objects owned by the claim to XML.

For a claim to be eligible for archiving, the server time must have reached the `Claim.DateEligibleForArchive` date:

- For more information on claim eligibly, see "Selecting Claims for Archive Eligibility" on page 668 in the *Configuration Guide*.
- For information on the archive work queue, and those claims that the archive process archived, excluded, or skipped, see "Archive Info" on page 160

After running the Archive work queue, Guidewire recommends that you update database statistics. The Archive work queue makes large changes to the tables. Updating database statistics enables the optimizer to pick better queries based on more current data. For instructions to gather database statistics, see "Configuring Database Statistics" on page 53.

See also

- "Archiving" on page 87 in the *Application Guide* – information on archiving claims, searching for archived claims, and restoring archived claims.
- "Archive Parameters" on page 39 in the *Configuration Guide* – discussion on the configuration parameters used in claims archiving.
- "Archiving Claims" on page 665 in the *Configuration Guide* – information on configuring claims archiving, selecting claims for archiving, and archiving and the object (domain) graph.
- "Archiving Integration" on page 285 in the *Integration Guide* – describes the archiving integration flow, storage and retrieval integration, and the `IArchiveSource` plugin interface.
- "Archive" on page 48 in the *Rules Guide* – information on base configuration archive rules and their use in detecting archive events and managing the claims archive and restore process.
- "Logging Successfully Archived Claims" on page 37 in the *System Administration Guide*.
- "Purging Unwanted Claims" on page 58 in the *System Administration Guide*.
- "Archive Info" on page 160 in the *System Administration Guide*.
- "Upgrading Archived Entities" on page 62 in the *Upgrade Guide*.

IMPORTANT To increase performance, most customers find increased hardware more cost effective than archiving unless their volume exceeds one million claims or more. Guidewire strongly recommends that you contact Customer Support before implementing archiving to help your company with this analysis.

Bulk Claim Validation

Code: `ClaimValidation`

The Bulk Claim Validation work queue creates work items to schedule loaded claims for validation.

This process can only be launched by using the `IClaimAPI.bulkValidate(loadCommandID)` API. The `loadCommandID` is an integer value that identifies the conversion batch process that imported the claims. The `loadCommandID` value is available through the `TableImportResult` object returned from a table import operation. For example:

```
ITableImportAPI.integrityCheckStagingTableContentsAndLoadSourceTables()
```

For more information, see "Determining the Validity of an Object" on page 82 in the *Rules Guide*.

Bulk Invoice Escalation

Code: `BulkInvoicesEscalation`

The Bulk Invoice Escalation work queue updates bulk invoices with a `status` of `Awaiting submission` and `send date` of the current date or later. This process transitions the status of each retrieved bulk invoice to `Requesting`. It also escalates all the checks associated with the invoice's items to **Submitting** status, as long as their `PendEscalationForBulk` fields are `true`. By default, this process runs daily at 6:35 a.m. and 6:35 p.m.

Bulk Invoice Submission

Code: `BulkInvoiceSubmission`

The `BulkInvoiceSubmission` work queue processes bulk invoice items for bulk invoice submission.

Processing each item consists of creating the placeholder check on that item's associated claim. After a bulk invoice is approved, the Bulk Invoice Submission batch process is run against the bulk invoice and creates a work item for each bulk invoice item. Then workers of the `BulkInvoiceSubmission` distributed work queue pick up each item and process that work item's bulk invoice item.

> **Note:** If a bulk invoice gets stuck in *PendingItemValidation* status and all workers have finished, run the Bulk Invoice Submission batch process again. This condition might indicate a misconfiguration, such as more than one thread configured for the `BulkInvoiceSubmission` work queue. This work queue must be configured with only one thread.

Bulk Invoice Workflow Monitor

Code: `BulkInvoiceWF`

Transitions the status of bulk invoices to either the `Awaiting submission` or `Invalid bulk invoice` status. Which status a bulk invoice transitions to depends on whether all the pending-approval checks were either approved or rejected. By default, this process runs every 30 minutes.

BulkPurge

Code: `BulkPurge`

Purges records through table updates. This process looks for `ClaimInfo` objects that are marked as retired and then traverses the domain graph to delete entities, presumable already retired, related to the retired claim.

Catastrophe Claim Finder

Code: `CatastropheClaimFinder`

Finds possible claims related to a catastrophe and creates an activity to review the claim to determine if it is associated with the catastrophe. By default, this work queue finds claims that meet the following conditions:

- The claim must not be already associated with a catastrophe.
- The claim loss date must be within the effective date range of the catastrophe.
- The loss type and loss cause of the claim must be listed in the coverage perils of the catastrophe.
- The claim must not have skipped or completed the `catastrophe_review` activity.
- The claim must not be retired.

You can modify this query by changing the `findClaims` method in `GWCatastropheEnhancement.gsx`. Access this enhancement in Studio at **configuration** → **Classes** → **gw** → **entity.**

Claim Contacts Calculations

Code: `ClaimContactsCalc`

Several contact fields are denormalized into different objects for performance reasons. This process recalculates denormalization fields in case they ever fall out of sync. The fields that this process updates are the following:

- Exposure.ClaimantDenorm
- Claim.ClaimantDenorm
- Claim.InsuredDenorm
- CheckPayee.PayeeDenorm
- Recovery.PayerDenorm

Only run this process if you encounter consistency check failures. The server must be at the maintenance run level to run Claim Contacts Calculations. You cannot run this process from the ClaimCenter user interface or schedule the process. You can only run Claim Contacts Calculations from the command line or by calling IMaintenanceToolsAPI.

See also

- "Using the Maintenance Run Level" on page 64
- "Running Batch Processes and Work Queues from the Command Line" on page 132
- "Starting or Querying Batch Processes Using Web Services" on page 84 in the *Integration Guide*

Claim Exception

Code: ClaimException

Claim Exception runs claim exception rules on claims that have been updated or had an exposure, transaction, claim contact, claim contact role, or matter updated since Claim Exception last ran. Claim Exception also runs claim exception rules on claims created since the last Claim Exception run. By default, Claim Exception runs every day at 2:00 a.m.

The SeparateIdleClaimExceptionMonitor configuration parameter controls whether Idle Claim Exception and Claim Exception processes run at separate times. If SeparateIdleClaimExceptionMonitor is false, then Idle Claim Exception and Claim Exception processes run concurrently to run claim exception rules for both recently modified and idle claims. SeparateIdleClaimExceptionMonitor is set to true in the default configuration.

For more information on claim exception rules, see "Claim Exception Rules" on page 59 in the *Rules Guide*.

Claim Validation

Code: ClaimValidation

The Claim Validation work queue creates work items to schedule loaded claims for validation.

Claim Health Calculations

Code: ClaimHealthCalc

The Claim Health Calculations process calculates health indicators and metrics for all claims that do not have any metrics calculated.

Guidewire provides claim exception rule **CER04000 - Recalculate claim metrics** that you can use to verify that ClaimCenter has created claim metrics, exposure metrics and claim indicators on every claim. See "Claim Exception Rules" on page 59 in the *Rules Guide*.

ContactAutoSync

Code: ContactAutoSync

The ContactAutoSync distributed work queue synchronizes Contact information between ContactManager and ClaimCenter. You can configure ClaimCenter to synchronize Contact information whenever a user updates a Contact, or you can use contactautosync to synchronize Contact information at a set time. Schedule this time in the scheduler-config.xml file. See "Scheduling Batch Processes and Distributed Work Queues" on page 143.

For more information on Contact synchronization, see “Synchronizing Contacts between ContactCenter and ClaimCenter” on page 39 in the *Contact Management Guide*.

Dashboard Statistics

Code: `DashboardStatistics`

Recalculates the statistics for user dashboards. By default, this runs every night at 1:00 a.m.

Data Distribution

Code: `DataDistribution`

Calculates the distribution of data in the ClaimCenter database.

You can configure and launch the Data Distribution process from the **Data Distribution** page. See “Data Distribution” on page 165.

Database Statistics

Code: `DBStats`

Calculates database statistics specific to ClaimCenter. You can also run separate processes to generate database statistics on staging tables only or only on tables that have had a configurable amount of data change.

For SQL Server, the `Auto Update Statistics` feature uses a specific algorithm (an undocumented counter) to decide when to update statistics. Usually, setting `Auto Create Statistics` and `Auto Update Statistics` to `True` is sufficient. If a recent operation changed a huge percentage of data or if performance has degraded due to bad query plans, then run the Database Statistics process explicitly.

You can configure and launch the Database Statistics process from the **Database Statistics** page.

See also

- “Configuring Database Statistics” on page 53
- “Database Statistics” on page 165

Encryption Upgrade

Code: `EncryptionUpgrade`

Upgrades encryption for entity fields.

ClaimCenter 6.0.8 provides encryption of sensitive data, such as tax IDs, on claim snapshots.

The `ClaimSnapshot` entity includes an `EncryptionVersion` integer column. This column stores a number representing the version of the IEncryption plugin used to encrypt the snapshot fields. ClaimCenter also stores the current encryption version. If the encryption metadata or plugin algorithm change, ClaimCenter increments this version number.

The Encryption Upgrade work queue recalculates encrypted field values for any `ClaimSnapshot` that has an `EncryptionVersion` less than the current encryption version. ClaimCenter decrypts the encrypted value by using the original plugin implementation. Then, ClaimCenter encrypts the value with the new plugin implementation. Finally, ClaimCenter updates the `EncryptionVersion` of the snapshot to mark it current.

You can adjust the number of snapshots that this process upgrades at one time by modifying the `SnapshotEncryptionUpgradeChunkSize` parameter in `config.xml`. A value of 0 for `SnapshotEncryptionUpgradeChunkSize` sets no limit to the number of snapshots upgraded at once.

For more information about the encryption plugin, see “Encryption Integration” on page 401 in the *Integration Guide*

Exchange Rate

Code: `ExchangeRate`

The Exchange Rate work queue uses an `ExchangeRateSetPlugin` to pull exchange rate data into ClaimCenter on a scheduled basis. You configure this batch process in `scheduler-config.xml`. By default, this exchange rate process is commented out and thus not enabled. If enabled, the `exchangerate` process is scheduled by default to run every morning at 2:00 a.m. Adapt this entry to configure your own batch process schedule. You must have an `ExchangeRateSetPlugin` activated first. For more information, see “Exchange Rates” on page 175 in the *Application Guide*.

Financials Calculations

Code: `financialscalculations`

The Financials Calculations process recalculates denormalized financial values. Several financial fields are denormalized into different objects for performance reasons. The Financials Calculations process recalculates denormalization fields in case they ever become unsynchronized. This process updates the following entities:

- `CheckRpt`
- `ClaimRpt`
- `ExposureRpt`

Each of these entities stores denormalized financials totals to improve performance when ClaimCenter displays the entity. These entities are kept current by the normal operation of ClaimCenter. However, issues in a custom configuration can cause the values to become unsynchronized and cause database consistency checks to fail. To help address such issues, Guidewire Support might ask you to run this batch process.

Never add new claims to ClaimCenter with the web service APIs while the Financials Calculations batch process is running. This applies to both `addFNOL` and `migrateClaim` methods.

This batch process can only be run from the command line while the server is at the maintenance run level.

See also

- “Using the Maintenance Run Level” on page 64
- “Running Batch Processes and Work Queues from the Command Line” on page 132

Financials Escalation

Code: `FinancialsEsc`

Submits any financial transactions with an `Awaiting submission` status and have passed their date requirements. By default, this runs every day at 6:05 a.m. and 6:05 p.m.

You must have purchased the financials extension to ClaimCenter.

See also “Issuing Scheduled Payments” on page 149.

Geocode Writer

Code: `Geocode`

The Geocode Writer batch process is the writer for the Geocode distributed work queue. This work queue runs periodically to update geocoding information on user contact primary addresses and account locations. The `UserContact` entity represents a ClaimCenter user.

By default, this schedulable process is disabled.

Group Exception

Code: `GroupException`

The Group Exception distributed work queue runs the group exception rule sets on all groups in the system. By default, this runs every day at 5:00 a.m.

For more information on group exception rules, see “Group Exception Rules” on page 62 in the *Rules Guide*.

Idle Claim Exception

Code: `IdleClaim`

The Idle Claim Exception distributed work queue runs the claim exception rules on claims that are idle. A claim is considered idle if the exception rules have not been run on it in the number of days specified by the `IdleClaimThresholdDays` configuration parameter. By default, Idle Claim Exception runs every Sunday at noon.

The `SeparateIdleClaimExceptionMonitor` configuration parameter controls whether Idle Claim Exception and Claim Exception processes run at separate times. If `SeparateIdleClaimExceptionMonitor` is `false`, then Idle Claim Exception and Claim Exception processes run concurrently to run claim exception rules for both recently modified and idle claims. `SeparateIdleClaimExceptionMonitor` is set to `true` in the default configuration.

For more information on claim exception rules, see “Claim Exception Rules” on page 59 in the *Rules Guide*.

ProcessHistoryPurge

Code: `ProcessHistoryPurge`

Purges batch process history data from the process history table. A large number of message history records in the database can slow performance when a user accesses the **Batch Process Info** or **Work Queue Info** system tools.

Before you run ProcessHistoryPurge, remove `WorkItem` and `InstrumentedWorkerTask` entities that reference `ProcessHistory`. Run the Purge Failed Work Items and Work Queue Instrumentation Purge processes to remove these entities.

The `BatchProcessHistoryPurgeDaysOld` parameter in `config.xml` specifies the number of days to retain batch process history. Set `BatchProcessHistoryPurgeDaysOld` to be `5` days greater than `InstrumentedWorkerInfoPurgeDaysOld` and not less than `35` days. For example, if `InstrumentedWorkerInfoPurgeDaysOld` equals `40`, then set `BatchProcessHistoryPurgeDaysOld` to `45`. This guarantees that referring work queue history records and failed work items are deleted by the Purge Failed Work Items and WorkQueueInstrumentationPurge processes. Work queue history records and failed work items that refer to a batch process history record must be deleted before ProcessHistoryPurge can delete the record.

Purge Failed Work Items

Code: `PurgeFailedWorkItems`

Purges failed work items from all distributed work queues.

Purge Message History

Code: `PurgeMessageHistory`

Purges old messages from the message history table. The `KeepCompletedMessagesForDays` parameter in `config.xml` specifies how many days a message can remain in the message history table before the Purge Message History process removes the message.

Purge Profiler Data

Code: `PurgeProfilerData`

Purges profiler data at regular intervals The `ProfilerDataPurgeDaysOld` parameter in `config.xml` specifies how many days to retain profiler data before the Purge Profiler Data process removes the data.

Purge Workflow

Code: `PurgeWorkflows`

Purges completed workflows after resetting any referenced workflows. The `WorkflowPurgeDaysOld` parameter in `config.xml` specifies how many days to retain workflow data before the Purge Workflow process removes the data.

This process executes `gw.processes.PurgeWorkflow.gs`. You can modify this Gosu class to customize the functionality of this batch process.

Purge Workflow Logs

Code: `PurgeWorkflowLogs`

Purges completed workflows logs. The `WorkflowLogPurgeDaysOld` parameter in `config.xml` specifies how many days to retain workflow logs before the Purge Workflow Logs process removes the logs.

This process executes `gw.processes.PurgeWorkflowLogs.gs`. You can modify this Gosu class to customize the functionality of this batch process.

Recalculate Claim Metrics

Code: `RecalculateMetrics`

The Recalculate Claim Metrics distributed work queue recalculates claim metrics for claims whose metric update time has passed.

See “Configuring Claim Health Metrics” on page 645 in the *Configuration Guide*.

ReviewSync

Code: `ReviewSync`

The ReviewSync process transmits completed reviews of service providers to ContactManager, creating a `ReviewSummary` object for each `Review`. Once `ReviewSync` submits a review, the process sets the `AddressBookUID` field on the `Review`, indicating that the `Review` has been transmitted.

For more information about service provider reviews, see “Service Provider Performance Reviews” on page 121 in the *Application Guide*.

Statistics

Code: `Statistics`

Recalculates business statistics. For example, this process updates each users open claim, activity, exposure, and matter counts. It also recalculates aging values on claims and exposures. By default this runs at 3 minutes past every hour.

TAccounts Escalation

Code: `TAccountEsc`

The TAccountEsc process updates the t-accounts of the payments on a future-dated check that has now reached its scheduled send date. After this update, the payment amount no longer contributes to **Future Payments**. Instead, the payment contributes to calculations that include payments scheduled for today, such as **Total Payments**, **Open Reserves**, **Remaining Reserves**, and **Available Reserves**.

The TAccountEsc process updates financials calculations when FinancialsEscalation or BulkInvoiceEscalation is not configured to run until later in the day, typically after the end of the business day. That way, calculations are current during the day that a check is scheduled to be sent, but the check is still editable since it has not yet been escalated.

The TAccountEsc batch process ensures that taccount balances and dependent calculated financials values are correct between midnight and the day's first scheduled execution of FinancialsEscalation and BulkInvoiceEscalation.

Schedule TAccountEscalation to run as close to just past midnight as possible and before FinancialsEscalation and BulkInvoiceEscalation. By default, this process runs daily at the beginning of the day, 12:01 a.m.

User Exception

Code: UserException

The User Exception distributed work queue runs the user exception rule set on all users in the system. By default, this runs daily at 3:00 a.m.

For more information on user exception rules, see "User Exception Rules" on page 62 in the *Rules Guide*.

Workflow

Code: Workflow

The Workflow distributed work queue wakes and runs workflow worker processes. See "Guidewire Workflow" on page 423 in the *Configuration Guide* for more information about workflows.

Work Queue Instrumentation Purge

Code: WorkQueueInstrumentationPurge

The Work Queue Instrumentation Purge process purges instrumentation data for work queues. The InstrumentedWorkerInfoPurgeDaysOld parameter defines how long to retain work queue instrumentation data. You can download work queue instrumentation data for a work queue by clicking **Download History** for the work queue on the **Work Queue Info** page. See "Work Queue Info" on page 152.

Scheduling Batch Processes and Distributed Work Queues

The ClaimCenter scheduler launches many batch processes, including writer processes for distributed work queues, according to a schedule defined in scheduler-config.xml. Access scheduler-config.xml from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources**. This file contains entries in the following format:

```
<ProcessSchedule process="process_code">
  <CronSchedule schedule_attributes/>
</ProcessSchedule>
```

The *process_code* is the process to run and *schedule_attributes* is a valid schedule specification. See "Defining a Schedule Specification" on page 144.

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

ClaimCenter 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 ClaimCenter scheduler uses the application server time for reference.

This topic includes:

Determining if a Batch Process Can Be Scheduled

Many batch processes, including distributed 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 ClaimCenter 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. ClaimCenter 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.
,	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.

These represent only some of the values that you can use for setting a schedule. The ClaimCenter scheduler is based on the open source Quartz Enterprise Job Scheduler version 1.2.3, and therefore uses the same values for the schedule attributes that Quartz uses.

The following examples show some common ways to use the `CronSchedule` element. For additional examples, refer to the Quartz documentation.

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 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.

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 ClaimCenter.

To determine the schedule in ClaimCenter

1. Log in to ClaimCenter 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 a work item, leading to further performance degradation.

You can use the internal ClaimCenter 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 ClaimCenter Scheduler to Ensure Batch Processes Do not Overlap

The internal ClaimCenter scheduler does not let you specify that one batch process must finish before another one begins. The ClaimCenter scheduler is purely a time-based scheduler. If you use the ClaimCenter scheduler, schedule the processes in your chain of nightly batch processes far enough apart so they are unlikely to overlap.

Using Your Own Scheduler to Ensure Batch Processes Do Not Overlap

Your organization may have its own scheduler for starting batch processes. If so, you can verify that the work of one batch process is complete before the next process in your chain of nightly batch processes begins. Use the `maintenance_tools` command-line tool or the `MaintenanceToolsAPI` web service to check the status of ClaimCenter batch processes.

See also:

- “Disabling the ClaimCenter Scheduler” on page 146
- “maintenance_tools Command” on page 171
- “Starting or Querying Batch Processes Using Web Services” on page 84 in the *Integration Guide*

Exclude Certain Batch Processes from Running During Your Nightly Batch Window

You may want to schedule some ClaimCenter batch processes to run periodically throughout the business day. For example, the default configuration of ClaimCenter 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.

Disabling the ClaimCenter Scheduler

You can choose to use your own mechanism for running ClaimCenter system processes. For example, you can use the ClaimCenter API or command-line utilities to run the processes, and you can use your own scheduling application to trigger their execution. If you do this, you might choose to disable the internal ClaimCenter scheduler. To disable the internal scheduler, set the `SchedulerEnabled` configuration parameter to `false`:

```
<param name="SchedulerEnabled" value="false" />
```

Using Events and Messaging with Batch Processes

You can use the event/messaging system to launch external processes upon completion of a batch process, including a distributed work queue writer. For example, you could write records to a file or database table as the batch process handles each transaction. After the batch process has handled all transactions, the data can be submitted to a downstream system. To coordinate this activity, use the `ProcessHistory` entity. As a batch process starts, ClaimCenter fires a `ProcessHistoryAdded` event. As the status of the process changes, ClaimCenter fires a `ProcessHistoryChanged` event. When ClaimCenter catches a `ProcessHistoryChanged` event, the `processHistory.CompletionTime` property contains the `datestamp` in a `datetime` object.

For more information about events and messaging, see “Messaging and Events” on page 139 in the *Integration Guide*.

Troubleshooting Batch Processes and Work Queues

This section discusses topics that are helpful in troubleshooting the management of batch processes.

Tuning Your Batch Process Schedule

To improve the performance of the batch operations, stagger the schedule so processes do not run at the same time. See “Scheduling Batch Processes and Distributed Work Queues” on page 143. You can also run these processes manually from the command-line.

Running Batch Processes from the Command Line

You can use the `maintenance_tools` command-line tool to run any of the batch processes. Run the following command:

```
maintenance_tools –password password –startprocess process_name
```

For example, to recalculate database statistics, you would run the following:

```
maintenance_tools –password password –startprocess dbstatistics
```

Monitoring Batch Processes

To check the status of a batch process, run the following command:

```
maintenance_tools –password password –processstatus process_name
```

You can also terminate a running process with the following command:

```
maintenance_tools –password password –terminateprocess process_name
```

Troubleshooting Distributed 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 work on 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 item’s `LastUpdateTime`, `CheckedOutBy`, and `Status`.

If a work queue is experiencing a large number of orphans, review log files to locate timeouts during processing. For example, a timeout might be caused by a worker waiting for an external server to return a value. If the log contains these type of timeouts, adjust the `progressinterval` value to give workers more processing time.

Sometimes, a problem inherent in the item itself causes processing of the item to fail. For example, an exception is thrown. In such cases, the worker stops processing the item and goes on to the next. The item becomes orphaned and the next worker attempts to process it. In this way, a distributed work queue attempts to process each item multiple times up to a limit set by the item’s configuration. If the work queue exceeds the item’s limit, ClaimCenter changes the item’s status to `failed`. Workers ignore `failed` items and no longer attempt to process them.

To remove failed work items, run the Purge Failed Work Items process.

Interactions Between ClaimCenter and Specific Processes

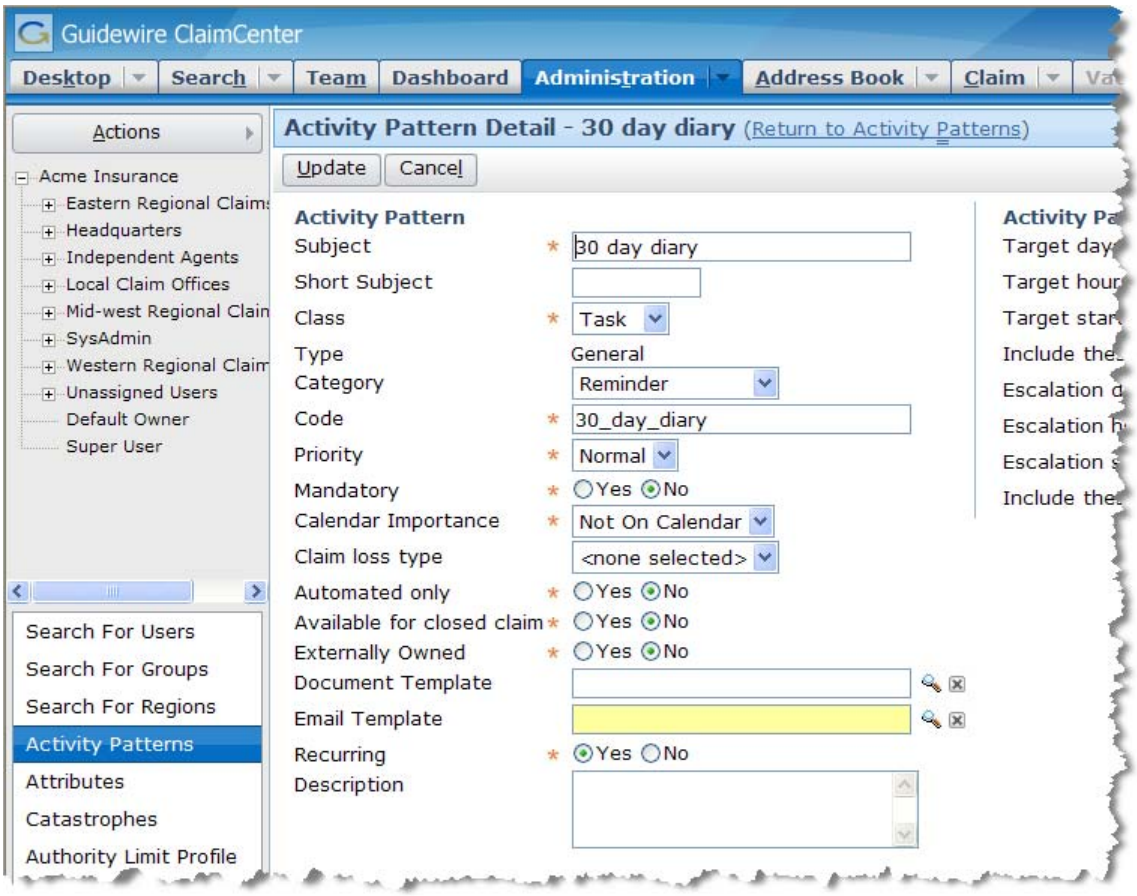
Activity Escalations

Activities in ClaimCenter can have an escalation date. The escalation date is the date on which ClaimCenter marks an open or overdue activity as requiring urgent attention. You can think of an escalation date as a deadline. ClaimCenter runs a scheduled process to find activities that need to be escalated due to the following criteria:

- The activity has a (non-null) escalation date.
- The escalation date has passed.
- The activity has not already been escalated.

If ClaimCenter finds an activity that meets this criteria, then it marks the activity as escalated and calls the **ActivityEscalationRules** in the **Exception** rule set to determine any actions.

Use the **Administration** console to define escalation values for an activity pattern. Select an activity from the **Activity Patterns** page and click **Edit**:



Typically, you set deadlines by using **Escalation Days.** However, you can set them in **Escalation Hours** or both depending on business practices.

If you set your deadlines only in days, then there is no reason to run activity escalation more than daily. However, if your deadlines are shorter, then run this process more frequently to take action on overdue activities in a timely manner.

Do not delete an activity pattern that is in production as there could be old activities tied to the pattern. Instead, change the activity pattern setting **Automated only** to **Yes** to prevent users from accessing it from the user interface.

By default, ClaimCenter runs `activityescalation` every 30 minutes. Change this schedule as needed.

Claim Exception Checking

ClaimCenter monitors claims for unusual or potentially concerning conditions (“red flags”) so that adjusters or supervisors can be alerted to look into it. This process is broken into two pieces:

- Selecting a list of claims to be evaluated.
- Consulting claim exception rules for each claim to find exceptions and take appropriate actions.

To be evaluated, a claim must meet one of the following two criteria:

- `Changed` – The claim must have changed in some *significant* way since the last time exception checking was run. Significant changes include editing the claim basics, adding or editing an exposure, and adding or changing a financial transaction. Adding notes or documents, for example, is not a significant change since exception rules would not normally evaluate the content of a new note or document.
- `Aging` – Some exceptions occur simply because the claim is aging without making progress. ClaimCenter handles this by evaluating any claim that has not been checked for reason #1 for the number of days defined by the `IdleClaimThresholdDays` configuration parameter.

Because running claim exceptions is processing intensive, run this during off hours. The two different criteria for selecting claims for evaluation can be run on different schedules. For example, you could use this feature to have ClaimCenter evaluate changed claims more frequently than aging claims.

To run separate schedules for changed claims and idle aging claims, set the `SeparateIdleClaimExceptionMonitor` parameter to `true`.

Issuing Scheduled Payments

Although many payments are intended to produce checks as soon as possible, ClaimCenter enables adjusters to schedule checks to be issued at a future date. This is very common, for example, while creating a schedule of recurring payments. The `financialsesc` process changes the status of these unissued payments from `Awaiting Submission` to `Submitting` so that they are sent to the financial system.

By default, this is set to run twice per day at 12:01 a.m. and 5:01 p.m. There is likely no need to run it more than daily since payments are scheduled for a certain day but not a certain time of day.

Calculating User Statistics

Users with the Supervisor role in ClaimCenter are able to view work activity statistics for the team by opening the **Statistics** window. The `statistics` process calculates work activity statistics. This process runs hourly by default. This process can be scheduled to run more frequently. However, monitor performance for possible negative impact if you change the schedule.

chapter 12

Using Server and Internal Tools

This topic discusses how to use the **Server Tools** and **Internal Tools** for administrative tasks.

This topic includes:

Using the Server Tools

Guidewire provides **Server Tools** to assist you with certain server and database administration tasks. This section contains the following topics:

Overview of Server Tools

Typically, users must have the `internaltools` permission to access the **Server Tools** pages. The **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 have access to the **Server Tools** pages. For more information about server modes, see "Understanding Server Run Levels and Modes" on page 62.

In addition to the **Server Tools**, there are a number of unsupported **Internal Tools**. Guidewire provides the **Internal Tools** for use during development only and does not support **Internal Tools** for production environments. See "Using the Internal Tools" on page 168.

Press `ALT+SHIFT+T` to display the **Server Tools** tab. At the top of each page in the title bar is a link that returns you to the main ClaimCenter interface. Alternatively, you can **Logout** from this page to exit ClaimCenter.

Batch Process Info

Use the **Batch Process Info** page to start and view information about ClaimCenter batch processes, including writer processes for work queues. This information includes the batch process **Name**, **Description**, **Status**, **Last Run** time, **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.

To start 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`.

Use the drop-down on the **Batch Process Info** page to filter the batch process list. You can set the filter to show **Any** processes, or only **Schedulable** or **Runnable** processes.

You can use the **Batch Process Info** page to view the history of a batch process. Select the batch process. The bottom pane provides **Chart** and **History** tabs. The **Chart** tab shows the execution time in seconds and the number of operations performed by the batch process over time. The **History** tab includes a table of records of past runs of the selected batch process. This **History** table includes the following information:

Column	Description
Started	The time that a batch process started.
Completed	The time that a batch process completed.
Ops	For batch processes that are work queue writers, **Ops** is the number of work items processed by a work queue. This number includes work items that failed.
Failed	For batch processes that are work queue writers, **Failed** is a counter that is incremented each time an exception is encountered while processing a work items. The work queue might attempt to process a failed work item multiple times. Therefore, the **Failed** and **Ops** numbers will not necessarily match the total number of work items.
Failure Reason	For batch processes that are work queue writers, **Failure Reason** is the reason that a work item failed processing.

You can configure this page to restrict the batch processes that can be started from this dialog. To change the configuration, edit the `ServerTools.pcf` page. Access `ServerTools.pcf` from the Guidewire Studio **Resources** pane by selecting **configuration** → **Page Configuration (PCF)** → **tools** → **ServerTools**. The `BatchProcessType` typelist in the *ClaimCenter Data Dictionary* lists the possible batch processes you can display.

For more information about batch processes and work queues available with ClaimCenter, see "Batch Processes and Distributed Work Queues" on page 134 and "maintenance_tools Command" on page 171.

Work Queue Info

Use the **Work Queue Info** page to control and view information associated with work queues. From this page, you can track work queues as they process information. Each work queue has both a writer and one or more workers. To work with this dialog, you must first select a **Work Queue**.

You can also click **Download** to download work queue information. This information includes a summary of the work queues and detailed information for specific work queues. The detailed information includes data for each worker by thread and by host, such as:

- how long the worker has been active
- how many items the worker processed
- throughput (items processed per minute of execution) per thread and per host
- start and end times for the worker
- last wake up time
- items processed per thread (cumulative, average, and maximum)
- uptime (cumulative, average, and maximum)
- execution time (cumulative, average, and maximum)

By default, ClaimCenter then shows you the statistics **By Writers** associated with the queue. Use the **By Workers** tab to view the workers associated with the queue.

The top-level columns have the following meanings:

Column	Description
Work Queue	Name of the work queue.
Available	Number of work items available for processing.
Checked Out	Number of work items checked out by workers.
Failed	Number of work items that failed during processing.
Workers 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 Workers**: Wake the workers by notifying them there are items to process. **Stop Workers**: Stops the workers currently processing the work queue. **Restart Workers**: Restarts the workers assigned to the work queue. **Download History**: Downloads the historical instrumentation data for the work queue. You can clear the instrumentation data for all work queues by running the Work Queue Instrumentation Purge process. See “Work Queue Instrumentation Purge” on page 143.

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.
Number of Items	The total work items in the queue regardless of status.
Execution Time	The number of minutes the process has been executing.
Available	The total number of available items in the queue.
Checked Out	The number of items checked out by workers for processing.
Succeeded	The number of items that completed successfully.
Failed	The number of items that failed.

The **By Workers** columns have the following meanings:

Column	Description
Hostname	The server on which the worker is running.
ID	The worker's process ID on the server.
Processed Items	Number of items processed by the worker.
Up For	The length of time the worker has been awake and processing work items.
Active	Specifies whether the worker is currently waiting for items to process.
Last Wake Up Time	The time at which the worker awoke.

The **Worker Runs** columns have the following meanings:

Column	Description
Start	The time at which the worker started its last processing run.
End	The time at which the worker ended its last run.
Success	How many work items the worker succeeded in processing.
Processed	The number of work items the worker processed.
Exception	The number of exceptions, if any, encountered during item processing.
Orphans Reclaimed	The number of orphaned work items ClaimCenter has adopted for processing.
Orphans Failed Count	The number of orphans permanently abandoned from the work queue.

Metro Reports

The **Metro Reports** page tracks the Metropolitan Reporting Bureau reports that ClaimCenter is processing. This is a cumulative reporting showing all the reports and their current status. You can filter the list of reports by date, status, and step. You can also use this page to resume processing of reports.

Management Beans

ClaimCenter includes management beans that represent different resources. You can use this dialog to view all and edit some of the attributes associated with these resources. The resources available from the dialog include:

Resource	Description
`com.guidewire.pl.system.monitor`	Tracks the sessions and connections associated with the ClaimCenter application server.
`com.guidewire.pl.system.cluster`	Provides information about the servers in a cluster. This bean is only available if you run the server in a clustered environment.
`com.guidewire.pl.system.configuration`	View all and edit some of the configuration values in the system. You must have the `soapadmin` permission to change values associated with management beans.
`com.guidewire.pl.system.cache`	View all and edit some cache attributes. You must have the `soapadmin` permission to change values associated with management beans.

Viewing and Changing Configuration Parameters

Using the `com.guidewire.pl.system.configuration` bean you can view all configuration parameters. You can edit some of these parameters from the **Management Beans** page. See "Application Configuration Parameters" on page 35 in the *Configuration Guide* for descriptions of configuration parameters.

Viewing and Changing Caching Configuration Values

For `com.guidewire.pl.system.cache`, you can set the `MaxCacheSpace` and `StaleTimeMinutes` values. The `MaxCacheSpace` value is the maximum amount of memory to use for the cache. The `StaleTimeMinutes` value is the number of minutes an object can exist in the cache without being refreshed. If an object is in the cache longer than `StaleTimeMinutes`, ClaimCenter refreshes the object entry. Use the “Cache Info” on page 158 page to monitor cache performance. Then, adjust cache values based on your analysis of that information. See “Understanding Application Server Caching” on page 70 for more information.

Guidewire Profiler

The Guidewire Profiler provides information about the runtime performance of code. The Guidewire Profiler collects information for specific sections of code that can be defined by Guidewire developers and configured, to some extent, for specific implementations.

The Guidewire Profiler does not collect memory usage statistics. You can use a third-party tool to gather memory usage and garbage collection information. See “Analyzing Server Memory Management” on page 75.

Guidewire Profiler Terms and Concepts

The following sample code introduces key concepts for understanding the Guidewire Profiler:

In `ProfilerTag.java`:

```
ProfilerTag MYTAG = new ProfilerTag("MyTag", "Sample code profiled for demonstration purposes");
```

In `MyCode.java`:

```
import gw.api.profiler.Profiler;
ProfilerFrame frame = Profiler.push(ProfilerTag.MYTAG);
try {
  myMethod(str);
} finally {
    Profiler.pop(frame);
}
```

Profiler Tag

Profiler tags represent sections of code that can be profiled by the Guidewire Profiler. A profiler tag is an alias for a piece of code in the Guidewire application for which you want to gather performance information.

Profiler tags are represented in the code by instances of the `gw.api.profiler.ProfilerTag` class. In this example, the profiler tag has the name `MyTag` and corresponds to the code invoked by the method `myMethod`. It is also given a description telling us that the code is just for demonstration purposes.

It is better to create a static final `ProfilerTag` and preserve it, rather than create one each time you need it. Creating the tag incurs some slight performance cost.

Profiler Frame

A profiler frame contains information corresponding to a specific invocation of profiled code, such as its start and finish times. When `push()` is called on the profiler stack, a profiler frame is created and pushed onto the stack. When `pop()` is called on the profiler stack, the profiler frame is removed from the stack, but its information is stored so as to be available for future examination. Profiler frames are represented in the code by instances of `gw.api.profiler.ProfilerFrame`.

Profiler Stack

A profiler stack stores profiling information for a specific thread. See “Entry Points” on page 156. A profiler stack implements the standard `push()` and `pop()` functionality of a stack. The `push` and `pop` correspond to the beginning and end, respectively, of a piece of code represented by a profiler tag. Thus, at any time the current contents of the profiler stack reflect all profiler tags whose code is currently being executed. Profiler stacks are represented in the code by instances of `gw.api.profiler.ProfilerStack`.

If a profiler stack has been initialized for the current thread, the call to `Profiler.push(ProfilerTag.MYTAG)` pushes a new frame with tag `MYTAG` on to that profiler stack. Otherwise, the call has no effect.

Similarly, `Profiler.pop(frame)` is just a pass-through to calling `pop()` on the profiler stack of the current thread.

Properties and Counters on a Frame

Profiler frames can hold user-defined properties and counters that provide more information about "what is going on". Consider this example:

```
frame.setPropertyValue("PARAMETER", str);
int ret = myMethod(str);
frame.setCounterValue("RETURN", ret);
```

When the profiler frame is popped off the stack, the frame contains information about which parameter was passed to `myMethod()` and the return value. Currently, properties and counters are used, among other things, to record SQL being executed, parameters to that SQL, row count returned, which rule is being executed, and so forth.

> **Note:** Exercise care when using this feature. Storing too much information will cause the display to become too cluttered, require more space for storage and, for long-running processes, hold on to too much memory at runtime.

Entry Points

The following entry points can initiate work on the application server:

Entry point	Description
Web	A user clicks around in a browser.
Batch process	Batch processes wake up at regular intervals, and can execute large queries. See “Batch Processes and Work Queues” on page 129.
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 Processes and Work Queues” on page 129.
Message destination	The process that sends messages out. See “Messaging and Events” on page 139 in the *Integration Guide*.
Web service	SOAP requests received by ClaimCenter. See “Web Services (SOAP)” on page 25 in the *Integration Guide*.
Startable plugin	You can create a plugin that is initialized at server startup and deinitialized at server shut-down. For example, a startable plugin can be registered as a listener on a JMS queue. See “Startable Plugins” on page 128 in the *Integration Guide*.

The Guidewire Profiler provides a unified means of configuration for profiling these entry points and also provides a unified way of accessing Guidewire Profiler data.

Configuring the Profiler

Configure the Guidewire Profiler on the **Configuration** tab of the Guidewire Profiler page.

You can choose to enable or disable profiling for specific entry points. Except for Web entry points, configuration information is stored in the database and is visible to all application servers in the cluster. Note that any changes will take some time to propagate through the cluster and it may take up to the cache stale time for a change to become visible.

The next time the entry point is started, the Guidewire Profiler checks whether profiling is enabled for the entry point. If profiling is enabled, profiling data is recorded in the form of a profiler stack. Multiple stacks are recorded if the initial thread spawns more and the developer profiles the spawned threads. Except for Web, this data is persisted to the database and can be later retrieved.

For Web entry points, profiling can only be enabled for the current session. On the **Configuration** tab, click **Enable Web Profiling for this Session** to enable profiling for the current session. All subsequent round-trips to the server will be recorded as a separate profiler stack. As opposed to stacks from other types of entry points, stacks from Web requests are not persisted to the database. Instead, stacks from Web requests are stored in the user session. Guidewire recommends that you enable the Web profiler only when it is needed. Enable the profiler, exercise the pages that you want to profile and then disable the profiler. Furthermore, log out after the profiler data has been analyzed to free memory used by the profiler.

Additional Tracing

You can enable additional tracing options. These options are quite expensive to compute. Guidewire recommends that you narrow down your performance issues first before trying these options.

Tracing option	Description
Individual Stacks	Only available for work queue entry points. When checked, this stores one stack per work item and does not roll up the data. Use caution with the **Individual Stacks** option as the amount of data to store could be unbounded. The **Individual Stacks** option is recommended for use when there are only a few items in the queue.
Stack trace tracking	Captures the Java stack trace and the PCF trace at the point where a query is executed.
Query optimizer tracing	Oracle only. This trace indicates how the database arrived at a specific execution path. This creates a trace file on the database server.
Extended query tracing	Oracle only. This trace tracks the entire parse-execute-fetch of a SQL statement including what it is waiting on, if anything. This creates a trace file on the database server.
DBMS Counter Threshold	Oracle only. If a database operation takes longer than this threshold, the DBMS counters will be captured at intervals equal to the value of the threshold. DBMS counters are stored in a cyclic buffer and the start of the buffer is lost for long operations.

IProfilerAPI

You can use the `IProfilerAPI` to configure the profiler through SOAP. In addition to being able to enable and disable profiling for the various entry points, the user can enable Web profiling on all subsequent sessions. This API does not provide the ability to profile a specific session or to profile active sessions. See “Profiling Web Services” on page 87 in the *Integration Guide* for more information.

Analyzing Profiler Data

The Guidewire Profiler provides the following views of profiler data.

View Type	Result
Stack Queries	Lists the queries that were executed by each stack. The first list shows the stacks in the profiled session. The second shows the queries executed in that stack. More details can be obtained by clicking on each tab.
Aggregated Queries	Lists all queries executed as part of the profiled session and some statistics about them, including number of times executed, average time, and more.
Search by Query	Searches the session for a query. This view enables you to determine the source of a particular query. Paste in a query, for example from the AWR report, and click **Search**.
Elapsed	Lists each frame in chronological order within its stack along with the time in seconds between when ClaimCenter pushed and then popped the frame.
Chrono	Lists each frame in chronological order within its stack along with the time in seconds between ClaimCenter creating the stack and pushing the frame.

View Type	Result
Group Frames	Lists frames in each stack aggregated by tag and presented in order of total aggregate time on the stack.
Group Stacks	Similar to **Group Frames**, except the Profiler aggregates the frames across all stacks in the session instead of by stack.
Rule Execution	Lists rules that fired during the session. If no rules fired during the session, the **Profiler Result** pane contains the message "No profiler stacks found". See "Generating a Rule Execution Report" on page 92 in the *Rules Guide*.

Downloading and Uploading Profiler Data

ClaimCenter serializes profiler data and stores it in the database in a different table for each entry point. You can download and upload profiler data as a ZIP file. To download profiler data, enable profiling for a particular entry point, select the **Profiler Analysis** tab and then select the entry point. Then, click **Download**.

To upload a ZIP file of profiler data, select **Profiler Analysis** → **Saved File**. Then click **Upload**.

Cache Info

The **Cache Info** page provides information about the application server cache usage for ClaimCenter. Use this page to review how well the cache is performing. The **Cache Info** page provides graphical representations of the application server cache, in **Cache Summary**, **Historical Performance** and **Cache Details** views. You can refresh any of these views by clicking **Refresh**.

The **Cache Summary** tab graphs include:

Graph	Description
Cache size	The memory used by the cache over time.
Cache hits and misses	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 ClaimCenter evicting an object because the cache was full and the number of missed caused by ClaimCenter evicting an object due to reaping.
Evict information	Information about cache evictions over time, including: • number of times no entry was found to evict when cache was full • number of evictions within active time when cache was full • number of evictions when cache was full • number of evictions due to reaping
Current age distribution	The number of objects of various ages in the cache.
Current cache contents for age 0	The types of objects present in the cache for age 0. These are objects that have just been added to the cache.

Click **Edit** to modify the maximum cache space and stale time parameters from the **Cache Summary** tab. If you change these parameters from the **Cache Summary** tab, the values you specify only apply to the application server node to which you are connecting and do not persist. If you restart the server, your changes are lost. To change these parameters and have the changes persist, edit the `config.xml` file. See "Understanding Application Server Caching" on page 70.

Click **Download** to download a CSV file containing information from the **Claim Summary** tab.

Click **Clear Global Cache** to clear the cache entirely of all entities. The cache always contains some objects to support an active application server.

The **Historical Performance** tab graphs include:

Graph	Description
Space retained	The memory used by the cache over the past couple days. The time shown is a much longer period than the cache size graph on the **Cache Summary** tab, which only displays the past 15 minutes.
Hits and misses	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 over the past couple days.
Miss %	The percentage of cache read attempts in which the object was not found in the cache over the past couple days.
Number of misses because item was evicted when cache was full	The number of misses over the past couple days due to ClaimCenter having evicted an object from the cache because the cache was full.

The **Cache Details** tab graphs include:

Graph	Description
Age distributions	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.
Cache contents	A number of graphs that show the percentage of types of objects in the cache over time.

Startable Plugin

The **Startable Plugin** page lists startable plugins that you have registered and enables you to manually start each plugin. For more information about startable plugins, see “Startable Plugins” on page 128 in the *Integration Guide*.

Set Log Level

Use the **Set Log Level** option to temporarily set the logging level for different logging categories. The logging level you specify persists until you change it or restart the server.

The logging categories available depend on your integrations and the settings in the `logging.properties` file. Access this file from the Guidewire Studio **Resources** pane under **configuration** → **Other Resources** → **logging**.

Each logging category has an associated level. You can change the runtime **Levels** value for each category using this dialog. If you restart the server, ClaimCenter does not retain your runtime settings. To make settings permanent, edit the `logging.properties` file. See “Configuring Logging” on page 35 for more information about how to configure logging and a description of the default logging categories.

View Logs

From the **View Logs** page you can select a log file to view, filter the log file for specific entries, and set the maximum number of lines to display.

To specify the location where the **View Logs** page checks for log files, specify the `guidewire.logDirectory` property in `logging.properties`. See “Specifying Location of Log Files for the View Logs Page” on page 36.

By default, ClaimCenter writes log files to `tmp/gwlogs/ClaimCenter/logs/`. You can configure log file locations and other properties for log files in the `logging.properties` file. See “Configuring Logging” on page 35.

Info Pages

The **Info Pages** provide screens that help Guidewire manage the server and database. Guidewire intends these pages for use by Guidewire Support, Integration Engineers, Database Administrators, and System Administra-

tors to diagnose existing and potential database-related performance problems. You can also use these pages to review the results of a load operation. You can access the following **Info Pages**:

- **Archive Info**
- **Domain Graph Info**
- **Consistency Checks**
- **Database Info**
- **Database Table Info**
- **Database Parameters**
- **Database Storage**
- **Data Distribution**
- **Database Statistics**
- **Oracle Statspack**
- **Oracle AWR**
- **Load History Information**
- **Load Integrity Checks**
- **Load Errors**
- **Upgrade Info**
- **Proximity Search Statistics**

Archive Info

Use the **Archive Info** page to view information about the archive. You must have the `ArchiveEnabled` parameter set to `true` in `config.xml` to view the **Archive Info** page.

The **Archive Info** page includes an overview, information about the archiving plugin, a summary of archiving information by data model version and details of each Archive work queue run.

Click **Refresh** to update the information on the **Archive Info** page. The **Refresh** button also refreshes the `IArchiveSource` plugin.

Click **Download** to download the archive information to an HTML report. This report includes the information shown on the **Archive Info** page.

Click **View Progress** to open the **Work Queue Info** page so that you can view the progress of the Archive work queue. To start an unscheduled run of the `archive` work queue, navigate to the **Work Queue Info** page. Then click the **Run Writer** button for the **Archive** work queue. For more information, see “Work Queue Info” on page 152.

The **Overview** includes the number of entities that have been archived and the number that have been skipped or excluded. Entities excluded from archiving are divided into those entities excluded due to rules and those excluded due to a failure. Click **Reset** to reset the total number of entities excluded due to rules or failure. Details for each skipped and excluded entity are provided at the bottom of the **Archive Info** page.

The **Archive Source Information** indicates when the **Archive Info** page and `IArchiveSource` plugin were last refreshed. To update the **Archive Info** page, click **Refresh**. The **Archive Source Information** also shows the availability of the store, retrieve and delete services. These are based on the `storeStatus`, `retrieveStatus` and `deleteStatus` of the `ArchiveSource.gs` plugin. Possible values for these services include:

- **Available** – The service is available.
- **Failure** – The last attempt to archive, restore or delete failed.
- **Manually** – The service has been manually flagged as unavailable.
- **Not configured** – The service has not been configured.
- **Not enabled** – Archiving has not been enabled.
- **Not started** – Archiving has not yet been started.
- **Queue** – The service is not available but allow queueing of user requests.

The **Archive Summary by Datamodel Version** shows archiving information by data model version. This information includes the earliest date and the latest date that entities were archived with each data model version. For each version the **Archive Info** page shows the number of entities that were archived, the number excluded due to rules, and the number excluded due to a failure. Each excluded category has a **Reset** button to reset the count of excluded entities. Click a data model version to see the **Archive Summary** page for that data model version. This page includes the **Reason for exclusion** for entities excludes by rules and entities excluded because of failure. For each reason, you can click **Reset All Items** to reset the count. On the **Archive Summary** page, you can specify a **Begin Time** and **End Time** to limit the information shown for the data model to a specific date and time range.

Note: All archiving and restoring operations also produce the usual log information.

See also

- "Archiving" on page 87 in the *Application Guide* – information on archiving claims, searching for archived claims, and restoring archived claims.
- "Archive Parameters" on page 39 in the *Configuration Guide* – discussion on the configuration parameters used in claims archiving.
- "Archiving Claims" on page 665 in the *Configuration Guide* – information on configuring claims archiving, selecting claims for archiving, and archiving and the object (domain) graph.
- "Archiving Integration" on page 285 in the *Integration Guide* – describes the archiving integration flow, storage and retrieval integration, and the `IArchiveSource` plugin interface.
- "Archive" on page 48 in the *Rules Guide* – information on base configuration archive rules and their use in detecting archive events and managing the claims archive and restore process.
- "Logging Successfully Archived Claims" on page 37 in the *System Administration Guide*.
- "Purging Unwanted Claims" on page 58 in the *System Administration Guide*.
- "Archive Info" on page 160 in the *System Administration Guide*.
- "Upgrading Archived Entities" on page 62 in the *Upgrade Guide*.

IMPORTANT To increase performance, most customers find increased hardware more cost effective than archiving unless their volume exceeds one million claims or more. Guidewire strongly recommends that you contact Customer Support before implementing archiving to help your company with this analysis.

Domain Graph Info

The **Domain Graph Info** page includes a **Graphs** tab that provides a DOT format representation of the domain graph and a **Warnings** tab to report issues with the graph. The DOT format is a means of showing object relationships in plain text. You can download the DOT format files and use a third-party tool, such as Graphviz, to view the graphs in a diagram format.

The Domain Graph

The domain graph represents the set of entities that relate to a root entity. The root entity is the main entity in the domain graph. In ClaimCenter, the root entity is the claim. Thus, the domain graph contains entities that relate to a claim, such as `Exposure`, `Coverage`, `Matter`, and other similar objects.

The domain graph defines the unit of work for object archiving. The unit of work for the archive process is a single instance of the graph, such as a single claim and all its associated entities. It is possible to associate certain entities with multiple graphs.

While the domain graph is central to the concept of archiving, ClaimCenter does not use it solely for archiving. ClaimCenter also uses the domain graph for the following:

- Claim purging

- Claim purges called during a cancel of the **New Claim** wizard

See also

- “Graph Validation Checks” on page 64
- “Archiving and the Domain Graph” on page 666 in the *Configuration Guide*
- “Purging Unwanted Claims” on page 58

Viewing the Graphs

If you view a graph as a diagram:

- The direction of the arrows in the diagram shows the direction of the "is owned by" relationship. Most of the time, this is also the direction of the foreign key. If the relationship is in the opposite direction to the foreign key, then the edge between the two entities is drawn in blue.
- A bold arrow from an entity to itself represents an edge foreign key.
- An open arrow from the array entity represents arrays and one-to-one relationships.
- Blue entities and links indicate extensions.

Warnings

The **Warnings** tab shows any violations of the following warning-level checks.

Check	Description
Domain graph entities refer only to administrative data and domain entities	All foreign keys from entities in the domain graph only reference other entities in the domain graph. Otherwise, foreign key violations can occur as ClaimCenter traverses the domain graph during archiving and purging processes.
Nothing outside the domain graph points to the domain graph	There must not be foreign keys from entities outside of the domain graph to entities in the domain graph. This prevents foreign key violations as ClaimCenter traverses the domain graph. This is not an outright failure because the archiving rule may prevent archival of such graphs anyway.
Null links cannot make node unreachable	ClaimCenter constructs the domain graph by looking at foreign keys, but the graph might not be a connected graph if a nullable foreign key is null. If enough links are null, the graph would become partitioned and the archiving or purging process would not be able to tag the correct entities. This check is a warning rather than one that prevents the server from starting because business logic might be in place that prevents the issue.

ClaimCenter provides warnings for these situations rather than preventing the server from starting because business logic may prevent the erroneous situation. The server also performs other graph checks while starting. If these checks fail, the server does not start. Because the server does not start, you cannot use the **Archive Graph Info** page to view errors detected by these checks. Instead, the server reports the error and prints the graphs in DOT notation. You can use this output with a graph visualization program to view the graphs. For more information, see “Graph Validation Checks” on page 64.

Download

Click **Download** to download information from the **Archive Graph Info** page. The downloaded ZIP file includes the following:

- `index.html` - a page that includes links to `construction.log`, `domain.dot` and `admin.dot` and shows any graph check warnings reported by ClaimCenter.
- `domain.dot` - a text file containing the domain graph in DOT format.
- `construction.log` - a text file that includes a log of how ClaimCenter constructed the graphs.
- `sorttable.js` - a javascript file containing javascript libraries used by `index.html` for showing graph warning information.

Consistency Checks

The **Consistency Checks** page enables you to view and run consistency checks on the ClaimCenter database.

The **View consistency checks definitions** tab lists the consistency checks available for each database table and provides a description of each check. You can search for consistency check types to see a list of all tables for which that consistency check is available. You can also search by table name to find the consistency checks related to that table. Most consistency checks operate on the specified table, but some checks, such as typelist table checks, operate on other tables. You can click **Download All** to download a ZIP file that contains HTML files describing all of the consistency checks provided with ClaimCenter. 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:

- Sort consistency checks by table name.
- Sort consistency checks by check name.
- View the SQL command of the consistency check. The SQL command retrieves a count of rows that violate the consistency check.
- View the SQL query used to identify rows that violate the consistency check. SQL queries to identify rows that violate consistency checks are not available for all check types.

From the `index.html` file only, you can do the following:

- Click a table name to view all consistency checks related to that table.
- Click a check name to view all tables that the consistency check runs against.

You can run consistency checks from the **Run consistency checks** tab of the **Consistency Checks** info page. You can specify to run consistency checks for all tables, for specific tables, or for a defined table group. To define table groups, see “Defining Table Groups” on page 45 in the *Installation Guide*. You can specify to run all checks or only specific consistency check types. You must specify one or more tables and one or more check types. When running consistency checks from the **Run consistency checks** tab, you can specify a **Description**. ClaimCenter prepends the description when you view the results to a standard description of the tables and checks. You can also specify the **Number of threads** to use to run the consistency checks.

The **Run consistency checks** tab also shows the results of prior consistency check runs. You can download the results of past consistency check runs. If there are consistency check errors, the results include SQL queries that you can use to identify records that violated the consistency check.

You can also run consistency checks with the `system_tools -checkdbconsistency` or `system_tools -checkdbconsistencyasbatch` command. 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 ClaimCenter Data Dictionary also lists available consistency check types. The typelist is `ConsistencyCheckType`.

For more information about running consistency checks with `system_tools`, see “system_tools Command” on page 173.

Database Info

The **Database Info** page details activity in the ClaimCenter database. The page provides totals for read/write requests and also shows the totals on deadlocks, timeouts, retries and surrenders in the database. This page can also you show you the tables in the database that have the most activity by SQL type — `INSERT`, `UPDATE`, and `DELETE`. You can sort the information in this page by selecting a heading to sort by.

Database Table Info

Use the **Database Table Info** page to locate index and key information for each table. These pages document the names of the ClaimCenter generated indexes and constraints. The following tabs appear on this page:

Tab	Description
Indexes by Table	Lists the indexes on a table and provides information about the key columns associated with it.
Primary Key Constraints by Table	Lists the primary key constraint on a table and provides information about the fields that reference the key.
Foreign Key Constraints by Table	Lists the foreign key constraints on a table and provides information about the table referenced by the key.
Max Row lengths	Displays the minimum and maximum row length in each table. Overly large row lengths in a database can lead to inefficiencies in data queries.

If your database is receiving integrity check errors or referential integrity problems, Guidewire Support might ask you to download the information from this page and provide it to them.

Database Parameters

The **Database Parameters** page displays information about the database configuration. A drop-down menu provides a list of database parameter types that you can view on this page. The following selections are available:

Tab	Description
Database and Driver	The versions of the database and its associated driver.
Database Connection Pool Settings	Connection pool settings as configured in `config.xml` if using ClaimCenter to manage the connection pool. See “Configuring Connection Pool Parameters” on page 47 for more information on these parameters. If you use the application server to manage the connection pool, then this page does not show connection pool parameters. Instead, tune the connection pool using the Administrative Console of the application server.
Guidewire Database Config	Guidewire-specific database configuration parameters. ClaimCenter 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. ClaimCenter 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 ClaimCenter is connected.
SQL Server Session Properties	SQL Server only. This view shows properties of the SQL Server session for the current connection with ClaimCenter.

You can click **Download Database Parameters Info** to download an HTML file containing all database parameters.

If you troubleshoot a database performance issue with Guidewire Support, Guidewire Support might ask you to send this parameter information as a reference. Integration consultants also use this data while tuning database performance.

Database Storage

The **Database Storage** page provides information about the space and memory taken up by the database on the server. The following tabs appear on this page:

Tab	Description
Database Space	Details about the amount of disk spaced taken up by the database.
Data Spaces	Lists the tablespaces (Oracle) or filegroups (SQL Server) taken up by the data and the amount of space taken and allocated by ClaimCenter.
TempDB Summary	Shows paging information for the database.
Indexes with High Fragmentation	Display average percentage of fragmentation for indexes in the database.
Index Physical Statistics	Lists the statistics about indexes on the physical table. Includes information such as minimum, average, and maximum record size. To change which **Tables and Indexes** the **Index Physical Statistics** tab displays, select a new one and click **Refresh**.
Tables and Indexes	Lists paging and allocation type of a table and its indexes. To change which table the **Tables and Indexes** tab displays, select a new table and click **Refresh**.

Download and save the database storage information right before and after an upgrade or other significant database change. This provides you with a point of reference you can provide to Guidewire Support if requested.

Data Distribution

Use the **Data Distribution** page to start a database distribution batch job. After the batch job completes, you can click **Refresh**. ClaimCenter lists the available distribution reports for download. ClaimCenter maintains each generated report until you **Delete** it.

Click **Download** to download a ZIP file containing the reports. The ZIP file contains HTML pages with the generated data.

To view the reports

1. Unzip the file into its own directory.
2. Locate the `index.html` file and double-click it to open it in a browser.
3. Use the links on the page to navigate through the distribution reports.

This page also lists data distribution reports for processes you start from the command line. To start the batch process from the command line, enter the following:

```
maintenance_tools -password password -startprocess datadistribution
```

Database Statistics

The **Database Statistics** page reports about out-of-date statistics in the database indexes, histograms, staging tables, and ClaimCenter tables.

You can generate database statistics for the entire database or for specific tables. Generating statistics for the entire database can be a very lengthy process on large databases. To specify tables on which to gather statistics, select **No** for the **Collect stats on all tables** option. Then select the checkbox next to each table for which you want statistics. You can also not select any tables and ClaimCenter gathers statistics information only from the database metadata.

After you generate statistics, ClaimCenter displays a report including 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 ClaimCenter tables. ClaimCenter 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 ClaimCenter 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 ClaimCenter 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.

Click **Download** to download the statistics report.

ClaimCenter provides database statistics generation designed specifically for how the ClaimCenter application and data model interact with the physical database. Generating database statistics from the database management system can potentially create statistics that cause ClaimCenter to select a bad plan for execution of SQL queries against the database. Therefore, always use ClaimCenter to generate database statistics, rather than using the statistics generation provided with the database management system.

Oracle Statspack

This option 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 using a tool such as SQL*Plus or SQL Developer. The **Oracle Statspack** page displays statspack snapshots you have created. A snapshot gives you database configuration and performance statistics for the duration you defined while creating the snapshot.

Oracle AWR

This option is available only if the database server is Oracle. The Oracle Automatic Workload Repository (AWR) is an upgrade of the information available with the Oracle statspack. Refer to Oracle documentation for details. You can use the **Oracle AWR** page to display AWR snapshots you define in the database.

Load History Information

The **Load History** page displays information about database load operations that completed in ClaimCenter. These load operations execute as you import data into the database. For example, loading data into the staging tables impacts the loader history information.

This page contains a summary view and a detail view. The summary view lists information about each specific load operation such as who called it, when, how long the operation took, any errors generated, and so forth. You can drill down into the details of an operation by clicking **View.**

The detail view shows on two tabs the **Steps** in the operation and the impact of the operation on **Row Counts**. You can drill down into the individual **Steps** by clicking on the step. Use the **Row Counts** page to quickly assess whether the amount of data that the operation loaded was the amount that you expected the operation to load.

See “Importing from Database Staging Tables” on page 423 in the *Integration Guide*

Load Integrity Checks

The **Load Integrity Checks** page reports on the SQL integrity checks that run as a database load operation executes. This page has two tabs: **View by Staging Table** and **View by Load Error Type**. To view the checks by table, select the former. To filter by error type, choose the latter.

For each integrity check, the **Load Integrity Checks** page lists the SQL query that the check performs. The **Load Integrity Checks** page also lists a description of the check and the associated load error type or staging table.

On both tabs, you can enable **Allow Non Admin References** or not. If you allow them, then ClaimCenter checks foreign key references to administrative tables (such as users and groups) on load. If this value is `false`, ClaimCenter does not check these references. The value is `false` by default.

See “Data Integrity Checks” on page 440 in the *Integration Guide*.

Load Errors

The **Load Errors** page displays errors generated by failed integrity checks. You can use this page to drill down through a table name to the specific error generated by a load operation. Errors relate to a particular staging table row. For each error, the **Load Errors** page shows:

- the table
- the row number
- the logical unit of work ID (LUWID)
- the error message
- the data integrity check (also called the *query*) that failed.

In some cases, ClaimCenter cannot identify or store a single LUWID for the error. For example, this might happen for some types of invalid ClaimCenter financials imports.

See “Data Integrity Checks” on page 440 in the *Integration Guide*.

Upgrade Info

The **Upgrade Info** page displays information about the automatic upgrade that runs upon server startup. For information on automatic upgrades see, “Understanding and Authorizing Database Upgrades” on page 50. You can use this page to see what steps were run by the upgrade and the impacts to row counts and storage information. From this page, you can also see the database parameters used during the upgrade.

Proximity Search Statistics

The Proximity Search Statistics page displays information about assignment proximity searches based on geocoded addresses performed in ClaimCenter.

Use the **Download** button to download a ZIP file containing proximity search statistics information. Guidewire Support might ask for this file if you experience poor performance with proximity searches.

Cluster Info

The Cluster Info page provides information on the clustered environment, if you have enabled clustering.

Using the Internal Tools

WARNING Guidewire does not support the **Internal Tools**. Guidewire provides these tools for use during development only. Guidewire does not support the **Internal Tools** for production environments. Use these tools at your own risk.

The **Internal Tools** page is only available if the server is in development mode. You can put the server in development mode by setting the JVM parameter `-Dgw.server.mode=dev`. Users with the `internaltools` permission can then access the **Internal Tools** pages by pressing `ALT+SHIFT+T` and selecting the **Internal Tools** tab. The **Admin** role has the `internaltools` permission by default. For more information about server modes, see "Understanding Server Run Levels and Modes" on page 62.

This topic contains the following:

- **Reload**
- **Update All Dates**

Reload

The **Reload** page is useful while you develop a configuration. From this page you can reload key configuration files into a running ClaimCenter 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, ClaimCenter writes the errors to the log.
Verify All PCF Files	Verifies the PCF files without reloading them.
Reload Web Templates	Reloads the entire ClaimCenter 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.

Update All Dates

You can use the **Update All Dates** page to offset all date fields in the database by an amount you specify. You can specify a positive or negative number of days for the offset. This is typically only done for demonstration data sets.

chapter 13

ClaimCenter Administrative Commands

This topic describes the administrative command-line utilities that ClaimCenter provides. Utilities for building ClaimCenter are described in "Build Tools" on page 71 in the *Installation Guide*.

ClaimCenter includes a number of commands to help you with administrative tasks on your server. The `ClaimCenter/admin/bin` directory contains these commands.

This topic includes:

fnol_mapper Command

```
fnol_mapper -help
fnol_mapper -password password [-server url] [-user user] {-input filename1  -mapping filename
            -property directory | -D name=value }
```

The `fnol_mapper` command accepts first notice of loss (FNOL) claim data in an XML file. A mapping class file contains a series of directives to transform the incoming FNOL data into a ClaimCenter `Claim` object. Then, ClaimCenter imports the object. The `fnol_mapper` command calls the `IClaimAPI.importClaimFromXML` method.

See the Gosu documentation for details. Generate Gosu documentation by navigating to `ClaimCenter/bin` and running the `gwcc regen-gosudoc` command.

See also

- "FNOL Mapper" on page 381 in the *Integration Guide*
- "Gosu Generated Documentation" on page 35 in the *Gosu Reference Guide*

fnol_mapper Options

You can use any of the following options with the `fnol_mapper` command. You must always supply the `-password` option.

`-D name=value`	Sets a Java system property.
`-help`	Prints a help message.
`-input filename`	Name of XML input file.
`-mapping filename`	Name of XML mapping file.
`-password password`	Specifies the *password* to use to connect to the server. ClaimCenter requires this option.
`-property directory`	Specifies directory containing all the property files, such as `logging.properties` and typelist property files.
`-server url`	Specifies the ClaimCenter host server url.
`-user user`	Specifies the *user* to run the process.

import_tools Command

```
import_tools -help
import_tools -password password [-server url] [-user user] [-ignore_all_errors]
             { { -import filename1, filename2 ... [{-dataset dataset | -output_csv filename |
             -output_xml filename}] | -D name=value |
             -privileges } [-ignore_all_errors] [-ignore_null_violations] }
```

Use this command to import new data into or update existing tables in the ClaimCenter database. You can only import data for valid entities or their subtypes. Guidewire supports this command for importing administrative data but not for importing other data into ClaimCenter. Instead, use staging tables or APIs to import other types of data into ClaimCenter.

Note: ClaimCenter does not fire any events related to the data you add or modify through this command.

import_tools Options

You can use any of the following options with the `import_tools` command. You must always supply the `-password` option.

Option	Description
`-D name=value`	Sets a system property.
`-dataset integer`	Specifies the *integer* representing the dataset to import. Datasets are ordered by inclusion. The smallest dataset is always numbered 0. Thus, dataset 0 is a subset of dataset 1, and dataset 1 is a subset of dataset 2, and so forth. To import all data, set this value to -1.
`-help`	Displays the command usage.
`-ignore_all_errors`	Causes the command to ignore any errors in a CSV file.
`-ignore_null_violations`	Causes the command to ignore violations of null constraints in import data.
`-import filename1, filename2, ...`	Imports administrative data from a comma-separated list of CSV or XML files.

Option	Description
`-output_csv filename`	If used with the `-import` option, outputs CSV to the specified file and then stops processing. ClaimCenter imports no data into the server. Use this command to convert XML input files to CSV.
`-output_xml filename`	If used with the `-import` option, outputs XML to specified file and then stops processing. ClaimCenter imports no data into the server. Use this command to convert CSV input files to XML.
`-password password`	Specifies the *password* to use to connect to the server. ClaimCenter requires the password.
`-privileges`	Rebuilds role privileges by deleting role privileges in the current database and importing the `roleprivileges.csv` file. The `import_tools` command looks first for a custom version of the `roleprivileges.csv` file in `ClaimCenter/modules/configuration/config/import/gen`. If `import_tools` does not locate a custom version in this directory, it checks for the default `roleprivileges.csv` file in `ClaimCenter/modules/cc/config/import/gen`.
`-server url`	Specifies the ClaimCenter host server url.
`-user user`	Specifies the *user* to run the process.

maintenance_tools Command

```
maintenance_tools -help
maintenance_tools -password password [-server url] [-user user] [-help] { -D name=value |
                  -getdbstatisticsstatements |
                  -markforpurge -claims claimnumber1, claimNumber2... | -markforpurge -file filename |
                  -scheduleforarchive -claims claimnumber1, claimNumber2... |
                  -scheduleforarchive -file filename |
                  -processstatus process | -startprocess process | -terminateprocess process |
                  -whenstats}
```

Use this command to start, terminate, or get the status of a ClaimCenter process. For a list of processes that you can start using `maintenance_tools`, see “Batch Processes and Distributed Work Queues” on page 134.

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
`-getdbstatisticsstatements`	Gets the list of SQL statements to update database statistics.
`-help`	Displays the command usage.
`-markclaimforpurge -claims claimnumber`	Marks an individual claim or comma-delimited list of claims for purging. See “Purging Unwanted Claims” on page 58.
`-markclaimforpurge -file filename`	Marks multiple claims for purging. The file contains a list of claims to purge. See “Purging Unwanted Claims” on page 58.
`-password password`	Specifies the administrative password. ClaimCenter requires a password to launch the maintenance tools.
`-processstatus process`	Gets the status of a batch process. For the *process* value, specify a valid process name or a process ID.
`-restore -claims claimnumber`	Restores one or more claims from the archive, with the supplied comment. This command uses the following format: `maintenance_tools.bat -restore comment -claims claimnumber -user user -password password` You can specify multiple claim numbers to restore separated by a comma.

Option	Description
`-restore -file filename`	Restores a group of claims from the archive, with the supplied comment. This command uses the following format: `maintenance_tools.bat -restore comment -file filename -user user -password password` The ascii file specified by *filename* must contain a list of claim numbers, separated by new lines.
`-scheduleforarchive -claims claimNumber1, claimNumber2...`	Flags an individual claim or comma-delimited list of claims for archiving. A separate process later archives claims flagged by this process. See "Archiving and Restoring Claims" on page 94 in the *Integration Guide*.
`-scheduleforarchive -file filename`	Flags multiple claims for archiving. The file contains a list of claim numbers to schedule for archiving. A separate process later archives claims flagged by this process. See "Archiving and Restoring Claims" on page 94 in the *Integration Guide*.
`-server url`	Specifies the ClaimCenter host server url.
`-startprocess process`	Starts a new batch process. For the *process* value, specify a valid process code. For a list of batch process codes, including work queue writer processes, see "Batch Processes and Distributed Work Queues" on page 134
`-terminateprocess process`	Terminates a batch process. For the *process* value, specify a valid process name or a process ID. Single phase processes cannot be terminated using this command. Single phase processes run in a single transaction, so there is no convenient place to terminate the process. The following batch processes are single phase processes that cannot be terminated: • AggregateLimitCalculation • DashboardStats • DataDistribution • ExchangeRate • FinancialsCalculation • Statistics • Table Import
`-user user`	Specifies the *user* to run the process.
`-whenstats`	Reports the last time ClaimCenter calculated statistics on the server.

messaging_tools Command

```
messaging_tools -password password [-server url] [-user user] [-purge date] [resume destinationID]
                [-resync -destination destinationID -claim claimID ]
                [-retry messageID] [-retrydest destinationID] [-skip messageID]
                [-statistics destinationID] [-suspend destinationID]
```

Use this command to manage a message destination from the command line. To manage a message destination from the command line, you must know its destination ID. The person who creates the message destination assigns this ID.

messaging_tools Options

You can use any of the following options with the `messaging_tools` command. You must always supply the `-password` option.

Option	Description
`-claim claimID`	The claim ID to resync. Used with the `-resync` and `-destination` arguments.
-destination *destinationID*	Specifies a specific message destination for resyncing. Used with `-resync`. Specify the claim with `-claim`.
`-password password`	Specifies the administrative password. You must specify a *password*.

Option	Description
`-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. If the purge tool succeeds in removing these messages without error, it reports "`Message table purged`". Since the number and size of messages can be very large, periodically use this command to purge old messages to avoid the database from growing unnecessarily.
`-resume destinationID`	Resumes the operation of the specified message destination.
`-resync`	Resyncs a claim with specified ID against a specific message destination. Use `-destination` and `-claim` to specify the destination and claim.
`-retry messageID`	Attempts to resend a message that failed. The message must be a candidate for retrying. A message is a candidate if the error at the destination system was temporary and the message destination has no automatic retry mechanism. For instance, if the record was locked and refused the update, the message would be a candidate for retrying.
`-retrydest destinationID`	Retries all retryable messages for a message destination.
`-server url`	Specifies the ClaimCenter host server url.
`-skip messageid`	Skips a message with the specified ID. If you mark a message as skipped, then ClaimCenter stops trying to resend the message. After you skip a message, you can not retry it.
`-statistics destinationID`	Prints the statistics for the specified destination.
`-suspend destinationID`	Suspends a message destination. Use this command if the destination system is going to be shut down or to halt sending while ClaimCenter processes a daily batch file.
`-user user`	Specifies the *user* to run the process.

system_tools Command

```
system_tools -help
system_tools -password password [-server url] [-user user] [-help] { -D name=value |
             -checkdbconsistency [tableselection checktypeselection] |
             -checkdbconsistencyasbatch [tableselection checktypeselection] |
             -daemons | -loggercats | -maintenance | -multiuser | -ping | -recalcchecksums |
             -reloadloggingconfig | -sessioninfo | -updateloggingleve l loggername loggingleve l |
             -verifydbschema | -version }
```

system_tools Options

You can use any of the following options with the `system_tools` command. You must always supply the `-password` option.

Option	Description
`-checkdbconsistency`	Checks the consistency of data in the database.The `-checkdbconsistency` option runs synchronously, waiting for completion before returning to the command line. The `-checkdbconsistencyasbatch` option runs consistency checks as an asynchronous batch process. Other than that distinction, the options and their arguments are the same. This tool has two optional arguments: `-checkdbconsistency` *`tableSelection checkTypeSelection`* The *`tableSelection`* argument can be specified as: • `all` – Run consistency checks on all tables. • *`table name`* – The name of a single table on which to run checks. • `tg.`*`table group name`* – The name of a table group. Table groups are defined in the `database` element of `config.xml`. For more information, see “Defining Table Groups” on page 45 in the *Installation Guide*. • `@`*`file name`* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with `tg.`, such as `tg.MyTableGroup`. You can combine table groups and individual table names in the same file. The *`checkTypeSelection`* can be specified as: • `all` – Run all consistency checks on the specified tables. • *`check name`* – The typecode of a single consistency check to run. • `@`*`file name`* – A file name with one or more valid consistency check names entered in comma-separated values (CSV) format. If you specify one optional argument, you must specify both. For more information, see “Checking Database Consistency” on page 51.
`-checkdbconsistencyasbatch`	Checks the consistency of data in the database using an asynchronous batch process. This option takes the same optional arguments as `-checkdbconsistency`. See the description for `-checkdbconsistency` and “Checking Database Consistency” on page 51 for details.
`-daemons`	Sets the server to the `DAEMONS` run level. For information about functionality available at various run levels, see “Understanding Server Run Levels and Modes” on page 62.

Option	Description
`-dbcatstats`	Used with no arguments, returns a ZIP file of database catalog statistics info for all the tables in the database. `-dbcatstats <regularTables stagingTables typelistTables>` Used with three arguments, returns a ZIP file of database catalog statistics info for the specified tables. Each of the arguments can be specified as: • `all`/`none` - Select all/none tables of this type, or • **<***table name***>** - The name of a single table of this type, or • **@<***file name***>** - A file name with one or more valid table names of this type entered in comma-separated values (CSV) format. For example, `-dbcatstats none none all` would return database catalog statistics information for all the typelist tables.You must specify either no arguments or three arguments while launching this tool. This option designates You can specify the target destination for the database catalog statistics ZIP file by adding the `-filepath` *filepath* parameter. If you do not provide a path, ClaimCenter uses the current directory. This process can take a long time, and it is possible for the connection to time out. If the connection times out while running this command, try reducing the number of tables to gather statistics on at a time by using the arguments shown previously. For information about configuring database statistics generation, see “Configuring Database Statistic Generation” on page 54.
`-help`	Displays the command usage.
`-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 “Understanding Server Run Levels and Modes” on page 62.
`-multiuser`	Sets the server to the `MULTIUSER` run level. For information about functionality available at various run levels, see “Understanding Server Run Levels and Modes” on page 62.
`-password` *password*	Specifies the administrative password. You must specify a *password*.
`-ping`	Pings the server to check if its active. The returned message indicates the server run level. The possible responses are: • `MULTIUSER` • `DAEMONS` • `MAINTENANCE` • `STARTING` For information about functionality available at various run levels, see “Understanding Server Run Levels and Modes” on page 62.
`-recalcchecksums`	Recalculates file checksums used for clustered configuration verification.
`-reloadloggingconfig`	Directs the server to reload the logging configuration file.
`-server` *url*	Specifies the ClaimCenter host server url.
`-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.
`-user` *user*	Specifies the *user* to run the process.
`-verifydbschema`	Verifies that the data model matches the underlying physical database.
`-version`	Returns the running server version, the database schema version, and configuration version.

table_import Command

```
table_import -help
table_import -password password [-server url] [-user user] { -D name=value |
             { [-batch] { -deleteexcluded | -updatedatabasestatistics | -integritycheckandload
             [-allreferencesallowed | -clearerror | -estimateorastats | -populateexclusion]
             -clearexclusion | -clearstaging | -outputerrors filename | -zonedataonly}
table_import -password password [-server url] [-user user] { -D name=value |
             {[-batch] { -deleteexcluded | -integritycheck [-allreferencesallowed | -clearerror |
             -populateexclusion | -updatedatabasestatistics ] -clearexclusion -clearstaging |
             -outputerrors filename }
```

table_import Options

You can use the `table_import` command to load data from staging tables into ClaimCenter.

You can use any of the following options with the `table_import` command. You must always supply the `-password` option. Before you can use this command, you must set the ClaimCenter server run level to `MAINTENANCE`. You can do this with the `system_tools` command. See "system_tools Command" on page 173.

Guidewire provides corresponding methods on `ITableImportAPI` for each of the `table_import` command options.

For more information, including detailed procedures to import data from staging tables, see "Importing from Database Staging Tables" on page 423 in the *Integration Guide*.

Option	Description
`-D name=value`	Sets a Java system property.
`-allreferencesallowed`	Allows references to existing rows in all source tables, including administrative tables such as users and groups. If there are rows in staging tables for `CheckGroup` or `CheckPortion`, you must set the `-allreferencesallowed` flag.

`-batch`	Runs the command in a batch process. This flag only applies with the following:
	`-integritycheck`
	`-integritycheckandload`
	`-populateexclusion`
	`-deleteexcluded`
	`-updatedatabasestatistics`

`-clearerror`	Clears the error table.
`-clearexclusion`	Clears the exclusion table.
`-clearstaging`	Clear the staging tables.
`-deleteexcluded`	Deletes rows from staging tables based on contents of exclusion table.
`-estimateorastats`	This parameter applies only to Oracle databases. It updates database statistics on the source tables with estimated row and block counts for the source tables and indexes at the beginning of load (`-integritycheckandload`).
`-help`	Displays the command usage.

`-integritycheck`	Validates the contents of the staging tables. You can optionally specify:
	`-allreferencesallowed`
	`-clearerror`
	`-populateexclusion`

`-integritycheckandload`	Validates the contents of the staging tables and populate source tables. You can optionally specify one of the following flags:
	`-allreferencesallowed`
	`-clearerror`
	`-estimateorastats`
	`-populateexclusion`
	`-zonedataonly`

`-messagesinks sinks, ...`	Deprecated. This flag does not do anything.
`-password password`	Specifies the administrative password. You must specify a *password*.

Option	Description
`-populateexclusion`	Populate the exclusion table with rows to exclude.
`-server url`	Specifies the ClaimCenter host server url.
`-updatedatabasestatistics`	Updates the database statistics on the staging tables. If you also specify the `-integritycheckandload` option, this option calculates the estimated row and block counts for the source tables. When running against Oracle, this command updates indexes before populating.
`-user user`	Specifies the *user* to run the process.
`-zonedataonly`	Sets the import to load zone data only. Used with the `-integritycheckandload` flag.

template_tools Command

```
template_tools -help
template_tools -password password [-server url] [-user user] [ -D name=value ]
               [ -working_dir name] { -convert_dir directory | -convert_file filename |
               -list_template | -import_dir objectsfile fieldsfile directory |
               -import_files objectsfile fieldsfile outfile | -validate_all | -validate_template id }
```

template_tools Options

You can use any of the following options with the `template_tools` command. You must always supply the `-password` option.

Option	Description
`-help`	Prints a help message.
`-convert_dir directory`	Converts templates in the specified directory to the new format.
`-convert_file filename`	Converts the specified template to the new format.
`-import_dir objectsfile fieldsfile directory`	Imports context objects and form fields into all the templates in the specified *directory*.
`-import_files objectsfile fieldsfile outfile`	Imports context objects and form fields into all the templates in the specified template *outfile*.
`-list_templates`	Lists all of the templates available for validation.
`-password password`	Specifies the administrative password. You must specify a *password*.
`-server url`	Specifies the ClaimCenter host server url.
`-user user`	Specifies the *user* to run the process.
`-validate_all`	Validates all the templates.
`-validate_template id`	Validates a single template.
`-working_dir directory`	Specify a directory so that relative paths can be used for file arguments.

usage_tools Command

```
usage_tools -help
usage_tools -password password [-server url] [-user user] [-D name=value | -count_exposures
            [-generate_csv | {-include_exposure_details | -include_monthly_details}] |
            -start_date startdate | -end_date enddate ]
```

usage_tools Options

You can use any of the following options with the `usage_tools` command. You must always supply the `-password` option.

Option	Description
`-count_exposures`	Count the total number of exposures.
`-D name=value`	Sets a Java system property.
`-end_date enddate`	Counts the total number of exposures up to and including this date. You can only specify this option with the `-count_exposures` option.
`-help`	Displays the command usage.
`-include_exposure_details`	Outputs all the exposure details. You cannot use this with the `-include_monthly_details` option.
`-include_monthly_details`	Includes the exposures per month in the output. You cannot use this with the `-include_monthly_details` option.
`-password password`	Specifies the administrative password. You must specify a *password*.
`-start_date startdate`	Counts the total number of exposures from this date forward. You can only specify this option with the `-count_exposures` option.
`-server url`	Specifies the ClaimCenter host server url.
`-user user`	Specifies the *user* to run the process.

zone_import Command

```
zone_import -help
zone_import -password password [-server url] [-user user] { -D name=value |
            {-import filename -country country [-clearstaging] [-charset charset]}
zone_import -password password [-server url] [-user user] { -D name=value |
            {-clearstaging [-country country]}
zone_import -password password [-server url] [-user user] { -D name=value |
            {-clearproduction [-country country]}
```

The zone_import command imports data from a specified file into zone data staging tables. To move data from the staging tables to the production tables, use the `table_import` command. The procedure is described later in this topic.

Guidewire expects that you import address zone data upon first installing the product and at infrequent intervals thereafter as you receive data updates.

Guidewire ships zone data files for the United States and Canada with each release. If you want data for a different country, speak to your Guidewire sales representative. Guidewire also includes smaller zone data sets for development and testing purposes. The zone data files are provided in `ClaimCenter\modules\pl\config\geodata`.

You can also create your own zone data files. The import tool uses the configuration from the `zone-config.xml` file to determine which data fields to import and what each field represents for each country. The `fileColumn` attribute for a `Zone` element indicates the positional value for that element within the zone data CSV file. For example, for zones in the United States, the `zone-config.xml` file includes:

```
<Zones countryCode="US">
  ...
  <Zone code="city" fileColumn="3" granularity="2">
  ...
```

This line specifies that the third comma-separated value in each line of a United States zone data file corresponds to the city. For more information about `zone-config.xml`, see “Configuring Zone Information” on page 476 in the *Configuration Guide*.

You can only import zone data for one country at a time. You can load zone data for multiple countries, but you must use a separate zone data file for each country and load each data set with a separate `zone_import` command.

If you create a zone data file, do not save the file to `ClaimCenter\modules\pl\config\geodata`. Doing so causes the server to fail while starting due to a module checksum failure. Save the file to `ClaimCenter\modules\configuration\config\geodata` or another location outside of the ClaimCenter modules.

To import the data file

1. Start the ClaimCenter server.
2. Set the ClaimCenter 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
   ```
   See “zone_import Options” on page 179 for a full list of options available with the `zone_import` command.
5. Clear existing zone data in production. This is useful if there is already zone data for the particular country that you are loading:
   ```
   zone_import -password password -country countrycode -clearproduction
   ```
6. Load zone data from the staging tables into production:
   ```
   table_import -password password -integritycheckandload -zonedataonly
   ```
7. Set the server run level back to `MULTIUSER`:
   ```
   system_tools -password password -multiuser
   ```

zone_import Options

You can use any of the following options with the `zone_import` command. You must always supply the `-password` option. Before you can use this command, you must set the ClaimCenter server run level to `MAINTENANCE`. You can do this with the `system_tools` command. See “Using the Maintenance Run Level” on page 64.

The same functionality supplied by this command is also available from a web service: `IZoneImportAPI`.

IMPORTANT For detailed procedures for importing zone data from staging tables, see “Importing from Database Staging Tables” on page 423 in the *Integration Guide*.

Option	Description
`-charset charset`	Character set in which the files are encoded. The default is UTF-8.
`-clearproduction`	Clears the zone data production tables. You can optionally specify the `-country` parameter to clear data only for that country.
`-clearstaging`	Clear the zone staging tables. You can optionally specify the `-country` parameter to clear data only for that country.
`-country countrycode`	Specifies the country.
`-D name=value`	Sets a Java system property.
`-help`	Displays the command usage.
`-import filename`	Imports zone data from the specified file. You must specify the `-country` value. You can optionally specify `-clearstaging` to clear data from the staging tables before importing from the file.
`-password password`	Specifies the administrative password. You must specify a *password*.

Option	Description
`-server url`	Specifies the ClaimCenter host server url.
`-user user`	Specifies the *user* to run the process.