

ClaimCenter Upgrade Guide

Release 6.0.8



Copyright © 2001-2013 Guidewire Software, Inc. All rights reserved.

Guidewire, Guidewire Software, Guidewire ClaimCenter, Guidewire PolicyCenter, Guidewire BillingCenter, Guidewire Reinsurance Management, Guidewire ContactManager, Guidewire Vendor Data Management, Guidewire Client Data Management, Guidewire Rating Management, Guidewire InsuranceSuite, Guidewire ContactCenter, Guidewire Studio, Guidewire Live, Gosu, Deliver Insurance Your Way, and the Guidewire logo are trademarks, service marks, or registered trademarks of Guidewire Software, Inc. in the United States and/or other countries.

This product includes information that is proprietary to Insurance Services Office, Inc (ISO). Where ISO participation is a prerequisite for use of the ISO product, use of the ISO product is limited to those jurisdictions and for those lines of insurance and services for which such customer is licensed by ISO.

This material is Guidewire proprietary and confidential. The contents of this material, including product architecture details and APIs, are considered confidential and are fully protected by customer licensing confidentiality agreements and signed Non-Disclosure Agreements (NDAs).

Guidewire products are protected by one or more United States patents.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org>).

Product Name: Guidewire ClaimCenter

Product Release: 6.0.8

Document Name: ClaimCenter Upgrade Guide

Document Revision: 05-February-2013

Contents

About This Document.	11
Intended Audience	11
Assumed Knowledge	11
Related Documents	11
Conventions In This Document	13
Support	13

Part I

Planning the Upgrade

1 Planning the ClaimCenter Upgrade.	17
Supported Starting Version.	18
Roadmap for Planning the Upgrade	18
Upgrade Assessment.	18
Preparing for the Upgrade.	20
Project Inception.	21
Design and Development	21
System Test.	22
Deployment and Support	22

Part II

Upgrading from 6.0.x

2 Upgrading the ClaimCenter 6.0.x Configuration	25
Overview of ContactCenter Upgrade	26
Obtaining Configurations and Tools.	26
Viewing Differences Between Base and Target Releases	27
Specifying Configuration and Tool Locations	27
Creating a Configuration Backup	30
Updating Infrastructure.	30
Deleting Target Configuration Module	30
Merging the ClaimCenter Configuration	30
Running the Configuration Upgrade Tool	31
Configuration Upgrade Tool Automated Step	31
Using the ClaimCenter 6.0.8 Upgrade Tool Interface	31
Configuration Merging Guidelines	37
Data Model Merging Guidelines.	38
Merging Contact Entities That Have Foreign Key or Array Extensions	40
Merging Other Files	40
About Display Properties	41
Upgrading Rules to ClaimCenter 6.0.8.	41
Rules Required for Key Features	42
Translating New Display Properties and Typecodes	43
Modifying PCF files, Rules and Libraries for Unused Contact Subtypes	43
Validating the ClaimCenter 6.0.8 Configuration	44
Using Studio to Validate Files	44
Starting ClaimCenter and Resolving Errors	44
Building and Deploying ClaimCenter 6.0.8	45

3	Upgrading the ClaimCenter 6.0.x Database	47
	Identifying Data Model Issues	48
	Verifying Batch Process and Work Queue Completion	49
	Purging Data Prior to Upgrade	49
	Purging Old Messages from the Database	49
	Purging Orphaned Policies from the Database	50
	Purging Address Correction Records	50
	Purging Completed Workflows and Workflow Logs	50
	Validating the Database Schema	50
	Checking Database Consistency	51
	Creating a Data Distribution Report	51
	Generating Database Statistics	52
	Creating a Database Backup	53
	Updating Database Infrastructure	53
	Preparing the Database for Upgrade	53
	Ensuring Adequate Free Space	53
	Disabling Replication	53
	Assigning Default Tablespace (Oracle only)	53
	Using Proper Clock Settings	53
	Handling Extensions	54
	Merging Extensions	54
	Reviewing Custom Extensions	54
	Reconciling the Database with Custom Extensions	54
	Setting Linguistic Search Collation for Oracle	55
	Using the IDatabaseUpgrade Plugin	56
	Running Custom SQL	57
	Eliminating Circular References	59
	Upgrading Archived Entities	62
	Disabling the Scheduler	64
	Suspending Message Destinations	64
	Configuring the Database Upgrade	65
	Configuring Column Removal on Oracle	65
	Configuring Index Creation Parallelism on Oracle	65
	Enabling Collection of Tablespace Usage and Object Size	65
	Enabling Oracle Logging	66
	Storing Temporary Sort Results in tempdb	66
	Adjusting Commit Size for Encryption	66
	Specifying Filegroup to Store Sort Results for Clustered Indexes	66
	Configuring Version Trigger Elements	67
	Starting the Server to Begin Automatic Database Upgrade	68
	Test the Database Upgrade	69
	Integrations and Starting the Server	69
	Understanding the Automatic Database Upgrade	69
	Version Trigger Descriptions	70
	Viewing Detailed Database Upgrade Information	72
	Dropping Unused Columns on Oracle	73
	Setting Claim Descriptions	73
	Final Steps After The Database Upgrade is Complete	74
	Backing up the Database After Upgrade	74

Part III

Upgrading from 5.0.x

4	Upgrading the ClaimCenter 5.0.x Configuration	77
	Overview of ContactCenter Upgrade	78
	Obtaining Configurations and Tools	78
	Viewing Differences Between Base and Target Releases	79
	Specifying Configuration and Tool Locations	79
	Creating a Configuration Backup	82
	Updating Infrastructure	82
	Deleting Target Configuration Module	82
	Removing Size Attribute from Integer and Money Datatypes	82
	Merging the ClaimCenter Configuration	82
	Running the Configuration Upgrade Tool	83
	Configuration Upgrade Tool Automated Steps	83
	Using the ClaimCenter 6.0.8 Upgrade Tool Interface	92
	Configuration Merging Guidelines	98
	Data Model Merging Guidelines	98
	Merging Contact Entities That Have Foreign Key or Array Extensions	100
	Merging Other Files	101
	Upgrading Rules	101
	About Display Properties	101
	Method Removed from Typekeys	101
	Merging PCF Files	101
	Merging Document, Email and Note Templates	108
	Upgrading Rules to ClaimCenter 6.0.8	108
	Rules Required for Key Features	109
	Translating New Display Properties and Typecodes	110
	Modifying PCF files, Rules and Libraries for Unused Contact Subtypes	111
	Converting Velocity Templates to Gosu Templates	111
	Validating the ClaimCenter 6.0.8 Configuration	114
	Using Studio to Validate Files	114
	Starting ClaimCenter and Resolving Errors	114
	Building and Deploying ClaimCenter 6.0.8	115
5	Upgrading the ClaimCenter 5.0.x Database	117
	Upgrading Administration Data for Testing	118
	Identifying Data Model Issues	120
	Verifying Batch Process and Work Queue Completion	121
	Purging Data Prior to Upgrade	121
	Purging Old Messages from the Database	121
	Purging Orphaned Policies from the Database	121
	Purging Address Correction Records	122
	Purging Workflow Logs	122
	Validating the Database Schema	123
	Checking Database Consistency	123
	Creating a Data Distribution Report	123
	Generating Database Statistics	124
	Creating a Database Backup	125
	Updating Database Infrastructure	125

Preparing the Database for Upgrade	125
Ensuring Adequate Free Space	125
Disabling Replication	125
Assigning Default Tablespace (Oracle only)	125
Using Proper Clock Settings	126
Handling Extensions	126
Merging Extensions	126
Reviewing Custom Extensions	126
Reconciling the Database with Custom Extensions	127
Creating Extensions to Preserve Data	127
Catastrophes	127
Deductible Fields	129
Reinsurance Status	130
Disabling Encryption for Upgrade Performance	131
Setting Linguistic Search Collation for Oracle	132
Using the IDatabaseUpgrade Plugin	133
Running Custom SQL	134
Eliminating Circular References	136
Upgrading Archived Entities	139
Disabling the Scheduler	141
Suspending Message Destinations	141
Configuring the Database Upgrade	142
Configuring Column Removal on Oracle	142
Configuring Index Creation Parallelism on Oracle	142
Enabling Collection of Tablespace Usage and Object Size	143
Enabling Oracle Logging	143
Storing Temporary Sort Results in tempdb	143
Adjusting Commit Size for Encryption	143
Specifying Filegroup to Store Sort Results for Clustered Indexes	144
Configuring Version Trigger Elements	144
Starting the Server to Begin Automatic Database Upgrade	145
Test the Database Upgrade	146
Integrations and Starting the Server	146
Understanding the Automatic Database Upgrade	146
Version Trigger Descriptions	147
Correcting Issues with Multiple Exposures on Incidents	158
Viewing Detailed Database Upgrade Information	160
Dropping Unused Columns on Oracle	161
Setting Claim Descriptions	161
Exporting Administration Data for Testing	162
Final Steps After The Database Upgrade is Complete	164
Running Key Batch Processes and Work Queues	164
Backing up the Database After Upgrade	165
6 Upgrading Integrations and Gosu from 5.0.x	167
Overview of Upgrading Integration Plugins and Code	167
Tasks Required Before Starting the Server	169
Tasks Required Before Deploying a Production Server	170
Tasks Required Before the Next Upgrade	171

Part IV

Upgrading from 4.0.x

7	Upgrading the ClaimCenter 4.0.x Configuration	175
	Overview of ContactCenter Upgrade	176
	Understanding Configuration Modules	176
	Obtaining Configurations and Tools	176
	Viewing Differences Between Base and Target Releases	177
	Obtaining Configurations	177
	Specifying Configuration Locations for 5.0 Upgrade Tool	177
	Creating a Configuration Backup	180
	Updating Infrastructure	181
	Upgrading the ClaimCenter 4.0 Configuration to 5.0	181
	Running the ClaimCenter 5.0 Configuration Upgrade Tool	181
	ClaimCenter 5.0 Upgrade Tool Automated Steps	181
	Updating ContactCenter Integration Files	190
	Deleting Target Configuration Module	190
	Remove Size Attribute from Integer and Money Datatypes	190
	Upgrading the ClaimCenter 5.0 Configuration to 6.0.8	190
	Specifying Configuration and Tool Locations	191
	Running the Configuration Upgrade Tool	192
	Configuration Upgrade Tool Automated Steps	193
	Using the ClaimCenter 6.0.8 Upgrade Tool Interface	201
	Configuration Merging Guidelines	207
	Data Model Merging Guidelines	208
	Merging Other Files	210
	Upgrading Rules	210
	About Display Properties	211
	Merging Classes (Libraries)	211
	Method Removed from Typekeys	211
	Merging PCF Files	211
	Merging Document, Email and Note Templates	221
	Merging Contact Role Constraints	221
	Merging ContactCenter Integration Files	221
	Upgrading Rules to ClaimCenter 6.0.8	222
	Rules Required for Key Features	223
	Importing Translation Properties	223
	Translating New Display Properties and Typecodes	224
	Modifying PCF files, Rules and Libraries for Unused Contact Subtypes	225
	Creating Modal PCF files for Custom Lines of Business	226
	Upgrading Assignment Rules	226
	Upgrading ISO Rules	226
	Upgrading ContactCenter Rules	226
	Converting Velocity Templates to Gosu Templates	227
	Validating the ClaimCenter 6.0.8 Configuration	229
	Using Studio to Validate Files	229
	Starting ClaimCenter and Resolving Errors	229
	Building and Deploying ClaimCenter 6.0.8	230
	Integrating ContactCenter with ClaimCenter	231
	Advanced Authentication to ContactCenter	231
8	Upgrading the ClaimCenter 4.0.x Database	233
	Upgrading Administration Data for Testing	234
	Identifying Data Model Issues	236

Verifying Batch Process and Work Queue Completion	237
Purging Data Prior to Upgrade	237
Purging Old Messages from the Database	237
Purging Orphaned Policies from the Database	237
Purging Address Correction Records	238
Purging Workflow Logs	238
Validating the Database Schema	238
Checking Database Consistency	238
Creating a Data Distribution Report	239
Generating Database Statistics	240
Creating a Database Backup	240
Updating Database Infrastructure	240
Enabling SQL Server READ_COMMITTED_SNAPSHOT Option	241
Preparing the Database for Upgrade	241
Ensuring Adequate Free Space	241
Disabling Replication	241
Assigning Default Tablespace (Oracle only)	241
Using Proper Clock Settings	241
Handling Extensions	241
Merging Extensions	242
Reviewing Custom Extensions	242
Reconciling the Database with Custom Extensions	242
Creating Extensions to Preserve Data	243
Deductible Fields	243
Reinsurance Status	244
Physical Property	245
Employment Class	245
Policy Property and Policy Vehicle	245
Disabling Encryption for Upgrade Performance	246
Setting Linguistic Search Collation for Oracle	246
Reviewing Data Model Changes	248
Default Size of mediumtext Columns	248
Exposure Fields Copied to Incident	248
Policy Data Model Changes	248
Injury Information Moved to New InjuryIncident Entity	250
New MessageHistory Entity	252
Address Entities Consolidated	252
Using the IDatabaseUpgrade Plugin	252
Running Custom SQL	253
Eliminating Circular References	255
Upgrading Archived Entities	258
Disabling the Scheduler	260
Suspending Message Destinations	260
Configuring the Database Upgrade	261
Configuring Column Removal on Oracle	261
Configuring Index Creation Parallelism on Oracle	261
Enabling Collection of Tablespace Usage and Object Size	261
Enabling Oracle Logging	262
Storing Temporary Sort Results in tempdb	262
Adjusting Commit Size for Encryption	262
Specifying Filegroup to Store Sort Results for Clustered Indexes	262
Configuring Version Trigger Elements	263

Starting the Server to Begin Automatic Database Upgrade	264
Test the Database Upgrade	265
Integrations and Starting the Server	265
Understanding the Automatic Database Upgrade	265
Version Trigger Descriptions	266
Correcting Issues with Multiple Exposures on Incidents	284
Viewing Detailed Database Upgrade Information	286
Dropping Unused Columns on Oracle	287
Setting Claim Descriptions	287
Exporting Administration Data for Testing	288
Final Steps After The Database Upgrade is Complete	290
Running Key Batch Processes	290
Backing up the Database After Upgrade.	291
9 Upgrading Integrations and Gosu from 4.0.x	293
Overview of Upgrading Integration Plugins and Code from 4.0.x.	293
Tasks Required Before Starting the Server.	294
Tasks Required Before Deploying a Production Server.	297
Tasks Required Before the Next Upgrade	299

About This Document

This document describes the process to upgrade an existing ClaimCenter installation from a previous version to the current version. This document does not describe data model configuration within a Guidewire version. For that information, see “Data Model Configuration” on page 179 in the *Configuration Guide*.

For information about new and changed features in ClaimCenter 6.0.8, see the *ClaimCenter New and Changed Guide*. For information about changes between minor releases of ClaimCenter, refer to the release notes for that release.

This topic includes:

- “Intended Audience” on page 11
- “Assumed Knowledge” on page 11
- “Related Documents” on page 11
- “Conventions In This Document” on page 13
- “Support” on page 13

Intended Audience

This document is intended for the following readers:

- System administrators
- Business users who want to upgrade to ClaimCenter 6.0.8 and understand the new features and changes in this release.

Assumed Knowledge

This document assumes that you are already familiar with the following topics:

- How to install systems with your operating system.
- How to install and administer the database of your choice.
- How to install and administer the application server of your choice.
- Using previous releases of ClaimCenter.

Related Documents

Guidewire has several Knowledge Base articles about the upgrade process. These articles are available on the Guidewire Resource Portal at <http://guidewire.custhelp.com>. Log in to the Guidewire Resource Portal, and select **Customer Support** → **Knowledge Base**. Then, under **Featured Support Categories** select **Upgrade**. You can then search for articles about specific issues and limit search results by product.

Also see the following Guidewire documents for further information:

ClaimCenter New and Changed Guide – Describes new features and changes to existing features in this version of ClaimCenter.

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 Reporting Guide – This document contains an overview of ClaimCenter reports. ClaimCenter uses InetSoft Style Report Enterprise Edition to provide a number of ClaimCenter-specific reports. This document describes how to install and configure the InetSoft application to work with ClaimCenter reports, as well as how to create and run reports.

ClaimCenter System Administration Guide – Provides guidance for the ongoing management of a ClaimCenter system. This document is intended to help system administrators monitor ClaimCenter, manage its security, and take care of routine tasks such as system backups, logging, and importing files.

ClaimCenter Application Guide – Introduces the application, explains application concepts, and provides a high-level view of major features and business goals of ClaimCenter. This is your first place to look when learning about a feature. This book is written for all audiences.

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.

Gosu Reference Guide – Describes the syntax of expressions and statements within ClaimCenter. This document also provides examples of how the syntax is used when creating rules. This document is intended for rule writers who create and maintain rules in Guidewire Studio.

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 Data Dictionary – Describes the ClaimCenter data model, including your custom data model extensions. To generate the dictionary, go to the `ClaimCenter/bin` directory and run the `gwcc regen-dictionary` command. To view the dictionary, open the `ClaimCenter/build/dictionary/data/index.html` file. For more information about generating and using the *Data Dictionary*, see the *ClaimCenter Configuration Guide*.

Conventions In This Document

Text style	Meaning	Examples
<i>italic</i>	Emphasis, special terminology, or a book title.	A <i>destination</i> 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 .
<code>monospaced</code>	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.
<code>monospaced italic</code>	Parameter names or other variable placeholder text within URLs or other code snippets.	Use <code>getName(<i>first</i>, <i>last</i>)</code> . <code>http://SERVERNAME/a.html</code> .

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

part I

Planning the Upgrade

Planning the ClaimCenter Upgrade

Upgrade your ClaimCenter installation frequently, in order to:

- Incorporate and use the new features of the new release.
- Reduce the number of versions to which you must upgrade to reach the current version.
- Remain compliant with your software license agreement.
- Continue to receive critical product support.

Your existing ClaimCenter installation is uniquely configured to meet the specific needs of your business. Thus, it is not possible to fully automate the upgrade process. Guidewire has taken steps to automate as much as possible, but there are always manual activities involved.

This topic assists you in planning the ClaimCenter upgrade to version 6.0.8. Read this topic before beginning an upgrade.

Also review the following topics before beginning the upgrade procedure:

- “What’s New and Changed in 5.0” on page 117 in the *New and Changed Guide* (if upgrading from 4.0.x).
- “What’s New and Changed in 6.0” on page 23 in the *New and Changed Guide* (if upgrading from 4.0.x or 5.0.x).
- “Release Notes Archive” on page 187 in the *New and Changed Guide* for changes to maintenance releases between major versions.
- “Upgrade Issues” in the ClaimCenter 6.0.8 release notes for issues specific to this release. If there are no upgrade issues specific to ClaimCenter 6.0.8, there is not an “Upgrade Issues” topic in the release notes.

Upgrade topics are presented together in parts of this guide according to your starting version.

If upgrading ClaimCenter 6.0.x, see the topics in “Upgrading from 6.0.x” on page 23.

If upgrading ClaimCenter 5.0.x, see the topics in “Upgrading from 5.0.x” on page 75.

If upgrading ClaimCenter 4.0.x, see the topics in “Upgrading from 4.0.x” on page 173.

This guide does not cover upgrading Guidewire Standard Reporting. See one of the following topics:

- If upgrading ClaimCenter 5.0.x, see “Upgrading from ClaimCenter 4.x to ClaimCenter 6.x” on page 43 in the *Reporting Guide*.
- If upgrading ClaimCenter 4.0.x, see “Upgrading from ClaimCenter 5.x to ClaimCenter 6.x” on page 59 in the *Reporting Guide*.

This topic includes:

- “Supported Starting Version” on page 18
- “Roadmap for Planning the Upgrade” on page 18
- “Upgrade Assessment” on page 18
- “Preparing for the Upgrade” on page 20
- “Project Inception” on page 21
- “Design and Development” on page 21
- “System Test” on page 22
- “Deployment and Support” on page 22

Supported Starting Version

You can upgrade ClaimCenter 4.0.x from version 4.0.3 or newer. You can upgrade any ClaimCenter 5.0.x or 6.0.x version.

If you are using a version prior to 4.0.3, first upgrade to the latest 4.0.x version. This version provides you with the latest database consistency checks for ClaimCenter 4.0.x. Consult Guidewire Services for assistance.

Roadmap for Planning the Upgrade

Before you begin an upgrade, plan and prepare for how an upgrade impacts both the business processes that rely on your Guidewire product and your organization as a whole. An upgrade requires commitment of time, personnel, and coordination among departments within your company.

Include these steps while defining your upgrade project:

- **Upgrade Assessment** - to plan the upgrade project.
- **Preparing for the Upgrade** - to prepare the environment and people for the upgrade project.
- **Project Inception** - to define the upgrade project.
- **Design and Development** - to implement the changes defined in scope for the upgrade project.
- **System Test** - to perform system, performance and regression testing for the upgrade, as well as perform deployment dry runs.
- **Deployment and Support** - to migrate the upgrade to production and provide necessary post-implementation support during the first few weeks after deployment.

Upgrade Assessment

The first step in planning an upgrade is to determine the impact of the upgrade on your implementation. During this phase of the project, you are:

- Performing an Opportunity Assessment.
- Analyzing the Impact on Your Installation.
- Creating Project Estimates.

These steps provide your users with business benefits and provide a clear understanding of resource requirements and the schedule you need to complete the project.

This phase typically take two to four weeks.

Performing an Opportunity Assessment

The *ClaimCenter New and Changed Guide* and the release notes describe new features available after upgrading ClaimCenter. As you review the new features, consider:

- How you might leverage these features into business benefits.
- How these features might help you improve your business processes.

Guidewire can also provide you with an evaluation copy of the new release, demonstrate new features for you, and help you understand opportunities these features can create for your business. Install the target version in a test environment. Take time to use the product to discover how to take advantage of the new features. Then, run some common scenarios for your company.

Analyzing the Impact on Your Installation

An upgrade might also impact your existing infrastructure, application configuration, and integration to other software. Make a careful evaluation of:

- Infrastructure Impacts
- Configuration Impacts
- Integration Impacts

Infrastructure Impacts

You might find it desirable or necessary to upgrade your hardware or software. If a major infrastructure element changes, such as a database version, factor that into your upgrade planning.

See the *Guidewire Platform Support Matrix* for system and patch level requirements for ClaimCenter 6.0.8. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at <https://guidewire.custhelp.com/app/resources/products/platform>.

Complete any infrastructure updates for your upgrade development environment before you begin the upgrade.

Configuration Impacts

Your company has uniquely configured ClaimCenter to meet its particular business needs. During an upgrade, configuration migration from the current version to the target version is your responsibility. Guidewire attempts to assist in this process by providing tools that locate and report back unique configurations in an installation. Additionally, The *ClaimCenter New and Changed Guide* and the release notes provide a reference for changes between each version and how these changes impact an installation.

Configuration migrations can take from days to weeks, depending on the complexity of the installation and resources available to perform the migration. If you choose to deploy new features in a release, implementing them can also impact the migration duration.

Pay special attention to areas in which your installation has significant custom configurations. For example, if you have extensively configured a user interface page, review its related files, data objects, and upgrade documentation for specific changes. Guidewire documentation is searchable. Search for key words or attributes to see if they changed between releases. While you do your review, keep in mind the following information:

- Is there current functionality your company uses that is absent from the target version?
- Is there new functionality in the target version that your company will use?
- Do you need to customize the new functionality or is it adequate as provided?
- If you have existing integrations, how are they impacted?

- Are there data changes between the current version and the target version?

Guidewire recommends first running the automated upgrade procedure in a development environment. This can help identify areas requiring work and give an indication of how much work is involved. These areas include:

Data Model: The ClaimCenter data model includes all ClaimCenter data entities and their relationships. The base data model changes between versions. If you have added extensions to objects in the base data model, the upgrade procedure migrates your extensions. However, if you have rules or integration code that reference your extensions, you are responsible for ensuring that they are correct after the upgrade.

User Interface: The automated process updates user interface customizations if possible. The update tool reports back which files it could not change. You are responsible for identifying unique customizations and migrating them into the new interface.

Other Configuration Files: In addition to user interface configurations, your installation probably includes unique system settings, security settings, and other configuration file settings. Generally, the upgrade preserves customizations to these files. However, changes in these files could require you to migrate to new configuration files manually. In addition, all icons and `.gif` files must be reapplied and references to them in PCF files redone. This is not done by the upgrade tool.

Rules: Your installation includes rules that govern its behavior at critical decision points. Your rules must reference only objects in the upgraded data model. Manual changes to rules are the only way to ensure correct references.

Gosu: Between versions, Guidewire deprecates or deletes some Gosu (formerly GScript) functions. You can not use removed functions. Manually remove deleted functions. Remove references to deprecated functions.

Integration Impacts

Your ClaimCenter implementation supports many integrations with external systems. An upgrade requires that you manually update or rewrite these integrations. Guidewire provides tools and sample integrations for newer releases. Parts of this document that describe upgrading from a prior major version include a topic about upgrading integrations and Gosu from the prior major version. The *ClaimCenter New and Changed Guide* and *ClaimCenter Integration Guide* can also help you estimate the changes required for each integration.

Creating Project Estimates

After completing the opportunity assessment and determining the impact of product changes on your implementation, you are able to define the scope of your upgrade project. Based on that scope you can identify some options for project staffing and schedule. If required, you can also produce a cost-benefit analysis for your upgrade project based on the scope and options defined.

Preparing for the Upgrade

Depending on your organization and implementation, you might need between two to eight weeks to prepare for the upgrade. This preparation work can be performed prior to or in parallel with the upgrade assessment and project inception phases of your upgrade project. Your main task is writing an upgrade specification. Other tasks in this phase include:

- Review results of the upgrade assessment and agree upon upgrade project scope.
- Review product documentation.
- Correct issues with database consistency checks.
- Ensure that your implementation documentation is up to date.
- Evaluate the skills and availability of internal resources.
- Identify and assign internal resources.
- Schedule and participate in “Delta Training” for the new release.

- Review and identify required changes to test scripts for the upgrade implementation.
- Acquire additional hardware and software to support infrastructure changes, if necessary.
- Set up a new environment for upgrade development.
- Set up separate branches for the upgrade in a source code control tool.

Writing an Upgrade Specification

Write an upgrade specification document that details the schedule, people, and equipment you expect to use during the upgrade. While writing the upgrade specification, answer the following questions:

- Does the ClaimCenter upgrade need software or hardware that your company must purchase? What is the expected lead time for a purchase at your company?
- Is the development and test infrastructure in place to ensure the new configuration meets company standards?
- Which old configurations are you upgrading?
- Which new features do you need to configure?
- Which integrations are impacted and to what extent?
- Does your staff have the appropriate knowledge of ClaimCenter, or is additional training required?
- What external support, if any, is required to execute the upgrade?
- If you are using external support, how are responsibilities divided among the team members?
- How does the company support production fixes while executing the upgrade initiative?
- If something goes wrong during the upgrade of the production environment, how do you recover?
- How does your company coordinate the production environment upgrade?
- Do you need to train your personnel on any aspects of the new system?
- Who supports the new configuration if users have questions?
- What type of documentation do you need for your new configuration?

To write an upgrade specification, take into account the testing and quality assurance your company requires. For each configuration upgrade and new feature in the configuration, define how that particular configuration will be tested and what criteria it must meet for acceptance.

Project Inception

Project inception typically takes between one to two weeks. During this phase:

- Review results of upgrade assessment..
- Finalize the project scope, resources and schedule.
- Ensure any required training has been performed by Guidewire Education.
- Make any necessary adjustments to assigned resources.
- Prepare the upgrade project plan.
- Hold workshops to review product functionality.
- Review the documented Guidewire upgrade steps and adapt them to your project.
- Perform project kick-off.

Design and Development

The design and development phase can take between two to three months depending on the extent of necessary changes. During this phase:

- Set up new environments for upgrade and performance testing.

- Define the system test plan.
- Update and creating system test scripts.
- Define user training requirements.
- Define your deployment and post go-live support plan.

System Test

Guidewire strongly recommends that you spend significant time testing your unique ClaimCenter configuration. Perform testing in a development environment, not a production environment. Follow the test plan you created in the planning phase. The length of time you spend testing depends on the complexity of your installation, but typically lasts between two and three months. During this phase:

- Perform system testing, including performance and stress testing. Application hardware configurations such as clustering, load balancing, and email servers must work as expected.
- Test your entire configuration: user interface, data model and extensions, business model and the rules that implement it, and integrations to external systems.
- Perform several dry-runs of the database upgrade against a current copy of the production database.
- Perform user acceptance testing.
- Finalize training materials and deliver training.
- Document step by step tasks and projected schedule for deployment weekend.
- Prepare the production environment for deployment.
- Finalize plans for post go-live support.

Deployment and Support

The final step in the upgrade roadmap is to deploy the upgraded implementation into the production environment. During this stage, coordinate within your company to ensure minimum interruption to business and sufficient time to qualify the new configuration in the production environment. Guidewire recommends that you perform the deployment over a weekend, with one to two weeks allocated for post go-live support. During this phase:

- Deploy the new configuration into production.
- Upgrade the production database.
- Verify the upgrade was successful.
- Provide heightened support.

Upgrading from 6.0.x

This part describes how to perform an upgrade from ClaimCenter 6.0.x to 6.0.8.

If you are upgrading from ClaimCenter 5.0.x, see “Upgrading from 5.0.x” on page 75 instead.

If you are upgrading from ClaimCenter 4.0.x, see “Upgrading from 4.0.x” on page 173 instead.

This part includes the following topics:

- “Upgrading the ClaimCenter 6.0.x Configuration” on page 25
- “Upgrading the ClaimCenter 6.0.x Database” on page 47

Upgrading the ClaimCenter 6.0.x Configuration

This topic describes how to upgrade the ClaimCenter and ContactCenter configuration from version 6.0.x.

If you are upgrading from a 4.0.x version, see “Upgrading the ClaimCenter 4.0.x Configuration” on page 175 instead.

If you are upgrading from a 5.0.x version, see “Upgrading the ClaimCenter 5.0.x Configuration” on page 77 instead.

This topic includes:

- “Overview of ContactCenter Upgrade” on page 26
- “Obtaining Configurations and Tools” on page 26
- “Creating a Configuration Backup” on page 30
- “Updating Infrastructure” on page 30
- “Deleting Target Configuration Module” on page 30
- “Merging the ClaimCenter Configuration” on page 30
- “Upgrading Rules to ClaimCenter 6.0.8” on page 41
- “Translating New Display Properties and Typecodes” on page 43
- “Modifying PCF files, Rules and Libraries for Unused Contact Subtypes” on page 43
- “Validating the ClaimCenter 6.0.8 Configuration” on page 44
- “Building and Deploying ClaimCenter 6.0.8” on page 45

Overview of ContactCenter Upgrade

The upgrade process for ContactCenter is almost precisely the same as for ClaimCenter. Follow the procedures in this guide to upgrade ContactCenter. Procedures specific to the ContactCenter upgrade are indicated as such. Some information in this guide is specific to ClaimCenter and does not apply to ContactCenter. However, the basic upgrade procedure is the same.

Obtaining Configurations and Tools

Configuration refers to everything related to the application except the database. This includes configuration files such as typelists and PCF files, the file structure, web resources, Gosu classes, rules, plugins, libraries, localization files, and application server files.

The upgrade process involves three configurations. This guide defines and refers to these configurations as **base**, **customer**, and **target**.

Base: The unedited, original configuration on which you based your customer configuration. The base configuration is included in directories within `/modules` other than `/configuration`.

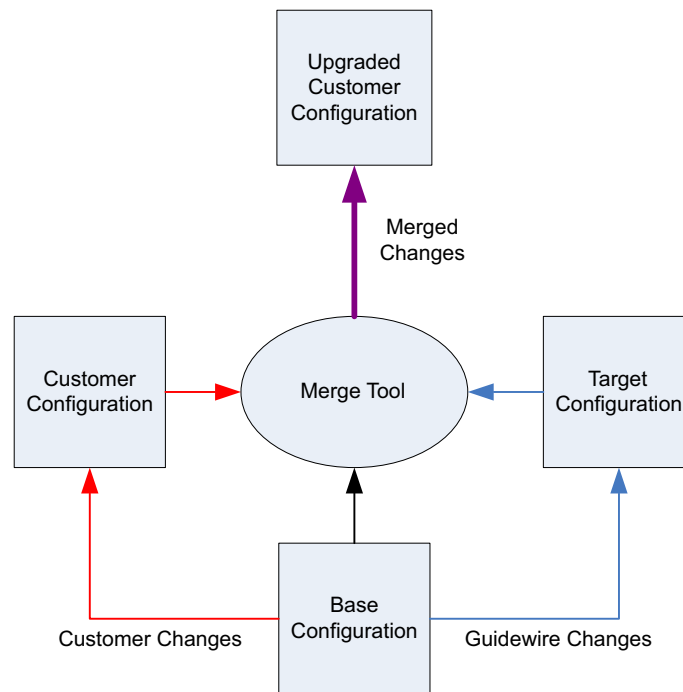
Customer: The configuration you are now using and will upgrade. This is the base configuration of the ClaimCenter version that you currently run with your custom configuration applied. The `/modules/configuration` directory contains custom changes.

Target: The unedited, original configuration of ClaimCenter 6.0.8 on which your upgraded configuration will be based. Guidewire grants you access to the Guidewire Resource Portal, from which you download the target configuration ZIP file. Unzip the target ClaimCenter 6.0.8 configuration into another directory. Download the latest patch release for the target version that you are downloading. Follow the instructions with the patch release to install it after you unzip the target version. Do not make any modifications to the target configuration prior to completing the configuration upgrade. Do not start Guidewire Studio for the target configuration until you have completed the configuration upgrade.

IMPORTANT Set all files in the base, customer, and target configurations to writable before beginning the upgrade.

The following figure shows how you use these configurations to create a merged configuration. The merged configuration combines your changes to the original base configuration (the customer configuration) and Guidewire

changes to the base configuration (the target configuration). The original base configuration provides a basis for comparison.



Viewing Differences Between Base and Target Releases

To view an inventory of the differences between the base release and the target release, download and carefully review the *Upgrade Diffs Report* from the Guidewire Resource Portal.

1. Open a browser to https://guidewire.custhelp.com/app/projectcenter/upgrades/diff_reports/.
2. Click ClaimCenter or ContactManager.
3. Click **Upgrade From** *base version*.
4. Click **Upgrade To** 6.0.8.
5. Download the three files into the same folder.
6. Open Report.html with a browser.

Specifying Configuration and Tool Locations

The ClaimCenter 6.0.8 Configuration Upgrade Tool depends on the following tools:

- **Text Editor:** An ASCII text editor you use to edit programs and similar files, for example, Notepad, WordPad or Textpad. This editor must not put additional characters in files, as Word does.
- **Merge Tool:** An editor which displays two or three versions of a file, highlights the differences between them, and allows you to perform basic editing functions on them. Also known as a “diff tool.” Examples include Araxis Merge Professional and Beyond Compare 3 Professional. Configure the merge tool to ignore end of line characters to reduce the number of potential false positives during the configuration upgrade step.

IMPORTANT The merge tool that you use must support three-way file comparison and merging. During the configuration upgrade, for some files you will need to compare three versions: the original base version, the new version and your customized version.

The Configuration Upgrade Tool needs the location of the ClaimCenter environment that you will upgrade. The tool stores all versions of files to be merged and merge results in a `/tmp` directory that it creates within the target environment. Define paths to the configuration and tools in the `/ClaimCenter/modules/ant/upgrade.properties` file of the target ClaimCenter 6.0.8 environment. Remove the pound sign, `#`, preceding each property to uncomment the line and then specify values. Use double backslashes in paths. For example, `C:\\ClaimCenter`. You do not need to use quotes for paths that include spaces.

The following properties are configurable in `upgrade.properties`.

Property	Description
<code>upgrader.priorversion.dir</code>	Path to the top-level ClaimCenter directory of the current customer environment. This directory contains <code>/bin</code> and other ClaimCenter directories.
<code>upgrader.editor.tool</code>	Path to an executable editing tool.
<code>upgrader.diff.tool</code>	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool, used for two-way merges. If your difference editor supports both two and three-way merges, you can use the same value for <code>upgrader.diff.tool</code> and <code>upgrader.merge.tool</code> .
<code>upgrader.merge.tool</code>	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool, used for three-way merges. The merge tool specified for <code>upgrader.merge.tool</code> must support three-way file comparison and merging. If your difference editor supports both two and three-way merges, you can use the same value for <code>upgrader.diff.tool</code> and <code>upgrader.merge.tool</code> . You might need to configure the display of your merge tool to show three panels.
<code>upgrader.merge.tool.arg.order</code>	The display order, from left to right, for versions of a file viewed in the difference editor tool specified by <code>upgrader.merge.tool</code> . By default, the tool displays, left to right, the versions of a file in this order: <code>NewFile OldFile ConfigFile</code> in which: <code>NewFile</code> is the unmodified target version provided with ClaimCenter 6.0.8. <code>OldFile</code> is the original base version. <code>ConfigFile</code> is your configured version. The order of these values controls the display order in the difference editor tool. If the tool displays just two versions, it uses the same relative order. By default, the display order places the old base version of a file in the center. The old base version is the common ground between the new uncustomized version and the old customized version. Guidewire changed the old base version to create the new target version, and you changed the old base version to create the customized version in your configuration. With the old base version in the center, you can incorporate both sets of changes to create a customized target version. The default order enables you to merge changes from the left and right to the center and save the merged version. If you use another difference editor, you might need a different order to achieve the same result. Samples are shown below for various difference engines: Araxis Merge Professional <code>upgrader.merge.tool.arg.order = NewFile OldFile ConfigFile</code> Beyond Compare 3 Professional <code>upgrader.merge.tool.arg.order = NewFile ConfigFile OldFile</code> P4Merge <code>upgrader.merge.tool.arg.order = OldFile NewFile ConfigFile</code> You might need to configure the display of your merge tool to show three panels.
<code>exclude.pattern</code>	A regular expression pattern for paths of files for the Configuration Upgrade Tool to mark as unmergeable. Typically, you use <code>exclude.pattern</code> to specify source control metadata files. Samples are provided in <code>upgrade.properties</code> for CVS and SVN.

Creating a Configuration Backup

Prepare the environment so that you can make a total recovery of the original installation if you run into problems during the upgrade.

Guidewire recommends that you track ClaimCenter configuration changes in a source code control system. Before upgrading, have a labeled version of your entire ClaimCenter installation. A labeled version is a named collection of file revisions.

As an even stronger precaution, make a backup of the same installation directories.

Updating Infrastructure

Before starting the upgrade, have the supported server operating systems, application server and database software, JDK, and client operating systems for the target version. See “Installation Environments Overview” on page 12 in the *Installation Guide* for supported versions for ClaimCenter 6.0.8.

Deleting Target Configuration Module

Delete all files and folders in the `modules\configuration` folder of the target ClaimCenter 6.0.8 configuration except `gwmodule.xml`. These files are included with new installations to help rapidly set up a QuickStart environment. You do not need these files to upgrade ClaimCenter, except `gwmodule.xml`, which is required by Studio.

During the manual configuration upgrade process, the Configuration Upgrade Tool copies files that you merge into the `configuration` module. If you do not delete the contents of the target `modules\configuration` folder, then the Configuration Upgrade Tool locates any file within the folder and considers it merged. However, the file is from the default target configuration rather than an actual merged file.

Merging the ClaimCenter Configuration

To upgrade your configuration, merge Guidewire changes to the base configuration with your changes. The *Configuration Upgrade Tool*, provided by Guidewire with the target configuration, facilitates this process.

The Configuration Upgrade Tool requires two tools: a merge tool such as Araxis Merge Professional or Beyond Compare 3 Professional, and a text editor. Configure the merge tool to ignore end-of-line characters to reduce the number of potential false positives during the configuration upgrade step.

IMPORTANT The merge tool that you use must support three-way file comparison and merging. During the configuration upgrade, for some files you will need to compare three versions: the original base version, the new version and your customized version.

The Configuration Upgrade Tool provides an interface that you use for the manual merge process.

Guidewire can provide guidance on using the Configuration Upgrade Tool in a multi-user environment. Knowledge Base article 571 describes distributing the merge work among users and using a source control management system for this process in a multi-user environment.

See the release notes for a description of changes in this release. Review key data model changes as these changes might impact customizations in your system. Release notes for prior versions are included in the “Release Notes Archive” on page 187 in the *New and Changed Guide*.

Running the Configuration Upgrade Tool

To launch the Configuration Upgrade Tool

1. Open a command window.
2. Navigate to the `modules/ant` directory of the target configuration.
3. Execute the following command:

```
ant -f upgrade.xml upgrade > upgrade_log.txt
```

You can specify any file to log messages and exceptions.

The Configuration Upgrade Tool first copies the modules of the base environment to a `tmp/cfg-upgrade/modules` directory in the target environment. The base environment is specified by the `upgrader.priorversion.dir` property in `ant/modules/upgrade.properties` in the target environment.

The Configuration Upgrade Tool then opens a user interface that you use to assist with the merging process.

Restarting the Configuration Upgrade Tool

The Configuration Upgrade Tool stores work in progress by recording which files you have marked resolved in the `accepted_files.lst` file. This file is stored in the `merge` directory of the target environment. You can close the interface and restart it later without losing your work in progress.

If you do want to start the upgrade over, use the `clean` command to empty the working directories.

```
ant -f upgrade.xml clean
```

WARNING If you empty the `tmp` directory after beginning to merge, you lose all completed merges that you have not resolved and moved into the target configuration directory.

Configuration Upgrade Tool Automated Step

The Configuration Upgrade Tool prepares for the manual configuration merge process by performing the following automated step.

Copying Display Properties Files into Target Configuration

This step copies `display.properties` files from the `locale` directories of the working configuration module to the target configuration module. The `display.properties` files do not require manual merging, so the Configuration Upgrade Tool copies them directly into the target configuration. If the file already exists in the target configuration, the tool skips the copy and logs a message. This is a precaution to make sure that the Configuration Upgrade Tool does not overwrite customized `display.properties` files if you run the tool again. If you are upgrading from a version prior to 6.0, this step does not copy the file because an earlier automatic upgrade step already performed the copy.

Using the ClaimCenter 6.0.8 Upgrade Tool Interface

After the Configuration Upgrade Tool copies files, the working area contains up to three versions of the same file:

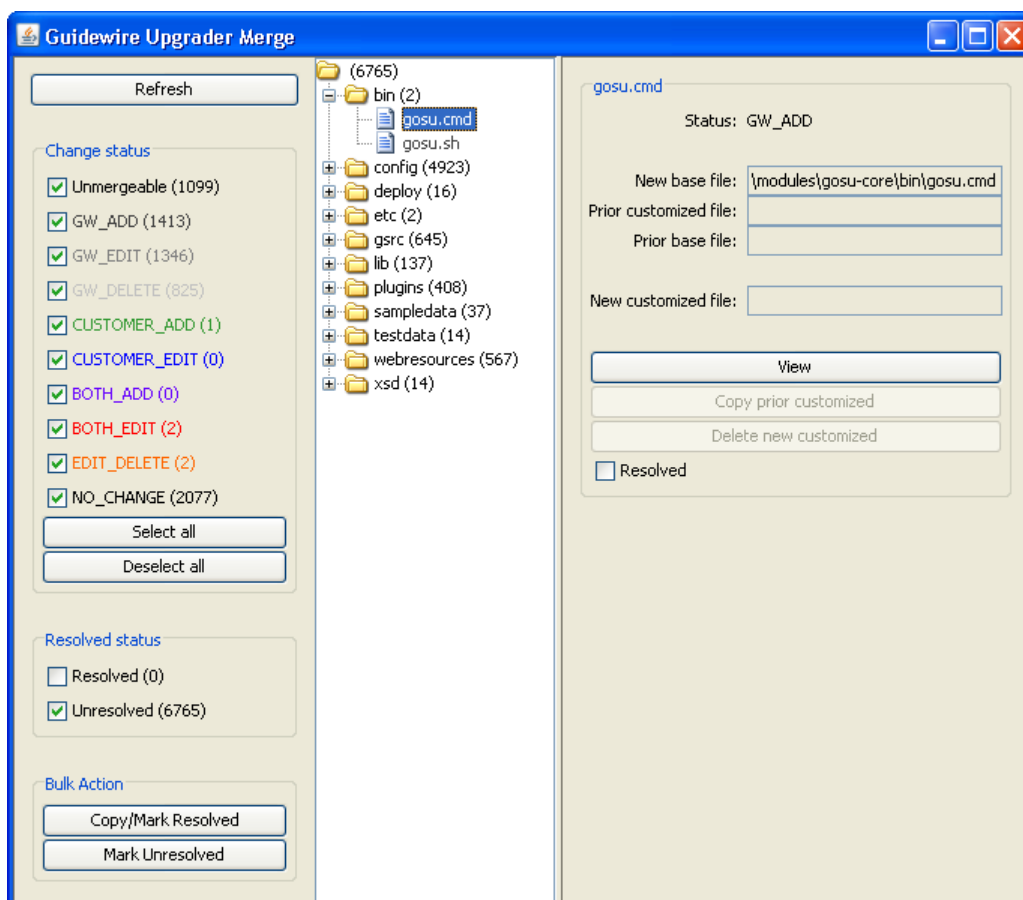
- the *customer* file
- the *base* file, from which you configured the customer file
- the *target* file, from ClaimCenter 6.0.8.

In the manual process of the upgrade, you decide whether to use one of these versions unchanged, or merge versions together. The Configuration Upgrade Tool provides a user interface to assist with the manual process. This interface has several important functions:

- It shows a complete list of all configuration files, organized into the module directory structure.
- It allows you to filter this list. You can, for example, view a list of all files that differ between the target version and your version. See “Change Status Filters” on page 33.
- It displays two or three versions of a file and their differences, using a merge tool you supply, such as Araxis Merge or P4Merge, defined in `upgrade.properties`.
- It lets you edit your file, incorporating changes from the other file versions, and save it.
- It lets you accept this merged version instead of one of the previous versions.
- It lets you edit the file after you have accepted changes from the merge using the text editor defined in `upgrade.properties`.

After you have accepted or merged all files that the Configuration Upgrade Tool displays, the merging process is complete.

The Configuration Upgrade Tool displays three panels. The center panel is a tree view of the files in the configuration, filtered by filter choices selected in the left panel. Files appear in the color of the filter that found them. As you select a file in the center panel, the right panel displays file information and buttons to perform actions on that file.



Filters

The left panel of the Configuration Upgrade Tool contains:

- **Refresh Button**

- Change Status Filters
- Resolved Status Checkboxes
- Bulk Action Buttons

Refresh Button

If multiple users are working in the same directory, each user can mark files as resolved. The **Refresh** button refreshes the resolved status of files shown in the Configuration Upgrade Tool for changes contributed by all users working in the same directory.

Change Status Filters

This table lists the change status filters that the Configuration Upgrade Tool displays in the left panel. Use the check boxes next to the filters to select one or any combination of change statuses to view. Use the **Select all** or **Deselect all** buttons to select or deselect all filters. The following table describes change status filters. The Guidewire Action column lists the change Guidewire has made to files matching a status filter since the prior version. The Your Action column lists the change to the file in your implementation:

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
Unmergeable	change format of file	any	file exists in a different format and thus cannot be merged with an old version	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file, in the new format, already exists in the target configuration.</p> <p>Files marked as unmergeable include rules and display properties files. Topics later in this guide describe upgrading these items.</p>
GW_ADD	add	none	file in target not in base	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file added by Guidewire already exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
GW_EDIT	edit	none	file in target differs from base	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file added by Guidewire already exists in the target configuration.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a comparison between the new Guidewire version and the original base version. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
GW_DELETE	delete	none	file in base not in target	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file deleted by Guidewire no longer exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
CUSTOMER_ADD	none	add	file in customer configuration only	<p>If you resolve the file, the Configuration Upgrade Tool copies the file to the target configuration module if the file has not been copied there already.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
CUSTOMER_EDIT	none	edit	file differs between customer and base configurations	If you resolve the file, the Configuration Upgrade Tool copies the file to the target configuration module if the file has not been copied there already.
			file unchanged between base and target configurations	Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a comparison between your custom version and the original base version. If you make changes, the tool prompts you to copy the file to the target configuration module.
BOTH_ADD	add	add	new file with matching name in both target and customer configurations (rare)	<p>You must either merge the two versions of the file or copy your prior version of the file into the configuration module before you can resolve the file.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a merge between your version and the Guidewire version. If you make changes, the tool prompts you to copy the merged file to the target configuration module.</p>
BOTH_EDIT	edit	edit	file changed in both customer and target configurations	<p>You must either merge the two versions of the file or copy your prior version of the file into the configuration module before you can resolve the file.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.merge.tool</code> in <code>upgrade.properties</code> to perform a three-way merge between your custom version and the updated Guidewire version. If you make changes, the tool prompts you to copy the merged file to the target configuration module.</p>

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
EDIT_DELETE	delete	edit	file changed from base in customer configuration and does not exist in target configuration	<p>If you resolve the file, the Configuration Upgrade Tool takes no action.</p> <p>Double-clicking the file opens your customized file and the original base file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code>. When you close the merge tool, the Configuration Upgrade Tool prompts you to copy the file to the target configuration module. If you are sure you want your customized version, you can click Copy prior customized to move the file to the configuration module of the target version.</p> <p>The EDIT_DELETE flag appears on a file when your configuration has a customized version of the file but Guidewire has deleted the file from that location. There are two possible reasons for this deletion. One reason is that Guidewire removed the file from ClaimCenter. The second reason is that Guidewire has moved the file to a different folder.</p> <p>If Guidewire has completely removed the file, review the <i>ClaimCenter New and Changed Guide</i>, release notes, and the Upgrade Diffs report for descriptions of the change affecting the deleted file. Then determine if you want to continue moving your customization to the new or changed feature. If not, then the customization will be lost.</p> <p>For the second scenario, find where the file has been moved by searching the target version. Move your customized file to the same location in the working directory and make sure to match any case changes in the file-name. When you refresh the list of merge files, the file now appears under the CUSTOMER_EDIT filter. You can now proceed with the merge. If you do not move the file over, you can instead perform the merge manually by opening both files and incorporating the changes.</p>
NO_CHANGE	none	none	file not changed from base configuration in either customer or target configurations	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file already exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>

Resolved Status Checkboxes

Beneath the change status filters are checkboxes to toggle the visibility of resolved and unresolved files. Use these checkboxes with the change status filters to specify which types of files you want visible in the center panel. For example, you could select **BOTH_EDIT** and **Unresolved** to see files edited in your configuration that have also been updated by Guidewire and are not yet resolved.

The purpose of the resolved status is to have a general idea of the progress you are making in the upgrade. The tool shows the resolved status of the current file (right panel) and the total number of resolved and unresolved files (left panel).

A resolved file is simply a file that you have marked resolved. It does not relate to whether file merging or accepting has occurred.

Bulk Action Buttons

The following buttons in The **Bulk Action** part of the left panel enable you to change the resolved status of a group of selected files:

- **Copy/Mark Resolved**
- **Mark Unresolved**

You can select either one or several files and directories before using these buttons. Use the CTRL key to select multiple files and directories. Selecting a directory selects all files within that directory. You can select all files that match the filters you set by selecting the top-level directory.

After you click **Copy/Mark Resolved**, the Configuration Upgrade Tool opens a dialog detailing the actions it is about to perform.

The tool copies files matching the **CUSTOMER_ADD** and **CUSTOMER_EDIT** filters to the target configuration module. If there is already a version of a file in the target configuration module, then the tool does not copy the file. A file would be there already if you edited the file and clicked **Yes** when the tool prompted you to copy the file to the configuration module.

The tool does not do any copying for files matching the **GW_ADD**, **GW_DELETE**, **GW_EDIT**, **NO_CHANGE**, or **Unmergeable** filters. Files matching **GW_ADD**, **GW_EDIT**, **NO_CHANGE**, or **Unmergeable** filters are already present in the target version. Files matching the **GW_DELETE** filter are not in ClaimCenter 6.0.8.

You can not bulk resolve multiple files that match the **BOTH_ADD**, **BOTH_EDIT**, or **EDIT_DELETE** filters. Files matching these filters require individual attention. Use the right panel of the Configuration Upgrade Tool to control merging, copying and resolving of these files.

Configuration File Tree

The center panel displays the configuration file tree. Files are color-coded to match filter colors. Files are shown one time, regardless of the number of configurations in which they exist. For information on which configurations a file exists in, select the file and view the right panel. The number of files in each directory that match the selected change status and resolved status filters is shown in parentheses.

File Details Panel

The right panel displays file details for the file you are currently examining, including:

- **Status:** the change status of the file. See “Change Status Filters” on page 33.
- **New base file:** The new version of this file supplied by Guidewire, typically in either the **cc** or **p1** module of ClaimCenter 6.0.8. If there is not a Guidewire version of this file, such as for **CUSTOMER_ADD**, **EDIT_DELETE** or **GW_DELETE** files, this field is blank.
- **Prior customized file:** The locally customized version of this file from the prior version. If there is not a customized version of this file, such as for **GW_ADD**, **GW_DELETE**, **GW_EDIT** or **NO_CHANGE** files, this field is blank.
- **Prior base file:** The base version of this file in the working directory. If there is not a Guidewire version of this file in the prior base version you are upgrading from, such as for **CUSTOMER_ADD** files, this field is blank.
- **New customized file:** The customized version of this file in the ClaimCenter 6.0.8 configuration directory.

The right panel fields are blank if you have multiple files selected.

File Details Panel Actions

The following buttons appear below the file details display in the right panel after you have selected a file:

- **View:** Opens the file in the editor specified in `upgrade.properties`. This button appears for files that are not customized and do not require merging, matching **GW_ADD**, **GW_EDIT**, or **GW_DELETE** filters. Only one of the **View**, **Edit** or **Merge** buttons displays, depending on the file change status.
- **Edit:** Opens the file in the editor specified in `upgrade.properties`. This button appears for custom files that do not require merging, matching the **CUSTOMER_ADD** or **EDIT_DELETE** filters.

- **Merge:** Opens the different versions of the file in the merge tool specified in `upgrade.properties`. This button appears for files that require merging, matching the **BOTH_ADD** or **BOTH_EDIT** filters.
- **Copy prior customized:** Copies the prior customized version of the file to the target configuration module. This button is enabled if there is a prior customized version of the file. So it is enabled for files matching **CUSTOMER_ADD**, **CUSTOMER_EDIT**, **BOTH_ADD**, **BOTH_EDIT**, or **EDIT_DELETE** filters.
- **Delete new customized:** Remove the customized version from the configuration module. This reverses the **Copy prior customized** button action. This button is disabled until you have copied a prior customized version of the file into the configuration module.
- **Resolved:** Check this box to label the file resolved. Use the **Resolved** checkbox in the right pane to change the status of a single file. Selecting the **Resolved** checkbox does not copy the file. Use the buttons above this checkbox to handle copying or merging of file versions. You must first unresolve a file before either using the **Delete new customized** action or reapplying changes or merges.

Accepting Files that Do Not Require Merging

The following filters show lists of files that normally do not require merging.

- **CUSTOMER_ADD**
- **CUSTOMER_EDIT**
- **GW_ADD**
- **GW_EDIT**
- **NO_CHANGE**
- **Unmergeable**

You can click the **Copy/Mark Resolved** button in the left panel to resolve groups of these files.

Merging and Accepting Files

Files matching the **BOTH_ADD** and **BOTH_EDIT** filters must be merged before being accepted. Your version must be reconciled with the Guidewire target or base version. In some cases, even if only a single version of the file exists, you might want to look at it before accepting it.

You can use the `pcf.xsd` file in `modules/cc/config/web/schema` of the target version to validate merged PCF files.

After you are satisfied with any changes, save the file. This saves the file in a temporary directory. When you close the editor or merge tool, the Configuration Upgrade Tool asks if you want to copy the file to the target configuration module. If you click **Yes**, the tool copies the file. If you click **No** (or **CTRL+N**), the tool cancels the popup without copying. The tool always moves files into the configuration module, except if a file is identical to the base or target version. In this case, the tool does not move the file.

Note: Do not edit a file version with **DO_NOT_EDIT** in its file name.

Configuration Merging Guidelines

First, work to generate the toolkit. The first milestone of an upgrade project is to generate a toolkit (by running `gwcc regen-java-api` and `gwcc regen-soap-api`) on the target release. To do this, you must:

- Complete the merge of the data model. This includes all files in the `/extensions` and `/fieldvalidators` folders.
- Resolve issues encountered while trying to generate the toolkit or start the QuickStart application server.

You can generate a toolkit even if you have errors in your enhancements, rules and PCF files.

Typical errors

- **Malformed XML:** The merge tool is not XML-aware. There might be occasions in which the file produced contains malformed XML. To check for well-formed XML, use free third-party tools such as Liquid XML, XML Marker, or Eclipse.
- **Duplicate typecodes:** As part of the merge process, you might have inadvertently merged in duplicate, matching typecodes.
- **Missing symmetric relationship on line-of-business-related typelists:** You might be missing a parent-child relationship with respect to the line-of-business-related typelist, as a result of merging.
- **References to GScript instead of Gosu,** such as in plugins/registry.

Once the toolkit has been generated, you can begin the work of upgrading integrations.

Second, after you can successfully generate the Java and SOAP APIs, work on starting the server.

In addition to the typical issues above, the server might fail to start due to cyclical graph reference errors. See “Identifying Data Model Issues” on page 48.

You can generate a toolkit even if you have errors in your enhancements, rules and PCF files, although the error messages will print upon server startup.

Once the server can start on the target release, you can begin the database upgrade process.

Continue with the remainder of the configuration upgrade work, which includes evaluating existing PCF files and merging in desired changes.

Data Model Merging Guidelines

From a purely technical standpoint, not addressing the need to incorporate new features, the following are a few guidelines for merging the data model.

Merging Typelists: Overview

There is no automated process to merge typelists. This is a part of the merge process using the Configuration Upgrade Tool. In general, merge typelists before PCF files.

Merge in Guidewire-provided typecodes related to lines of business and retire unused typecodes that you merge in. If you do not include these typecodes, you will have errors in any enhancements, rules, or PCF files that reference the typecode. This also simplifies the process for future upgrades as there will be fewer added line of business typecodes to review.

Pay particular attention if any Guidewire-provided typecodes have the same typecode as a custom version. In this case, modify one of the typecodes so they are unique. Contact Guidewire Support for details.

The Configuration Upgrade Tool displays most typelists you have edited in the CUSTOMER_EDIT filter. If your edits are simply additional typecodes, accept your version.

Use Guidewire Studio to verify PCF files, enhancements, and rules to identify any issues with the files and rules that reference typelists.

Merging Typelists: Simple Typelists

Merge in new typecodes from the target version, ClaimCenter 6.0.8. If you do not merge the new typecode, you will have errors in any enhancements, rules, or PCF files that reference the typecode. If you do not want to use a new typecode, retire the typecode by setting the `retired` attribute to `true`.

In the following example, the target version of ClaimCenter (shown on left) introduced the `debrisremoval` typecode.



To avoid errors in enhancements, rules, and PCF files that reference the `debrisremoval` typecode, merge the typecode. If you do not want to use the new typecode, merge it into the target file and retire it by setting the `retired` attribute to `true`, as shown below.



Merging Typelists: Complex Typelists

A typecode can reference typecode values from another typelist using the `<category>` subelement. If a new typecode references an existing typecode, do not merge the new typecode unless the referenced typecode is retired. Otherwise, you are defining a new relationship. If the referenced typecode is also new, merge in both typecodes. If you do not want to use a new typecode, set the `retired` attribute for the typecode to `true`. The following table summarizes how to handle merging new typecodes that reference other typecodes:

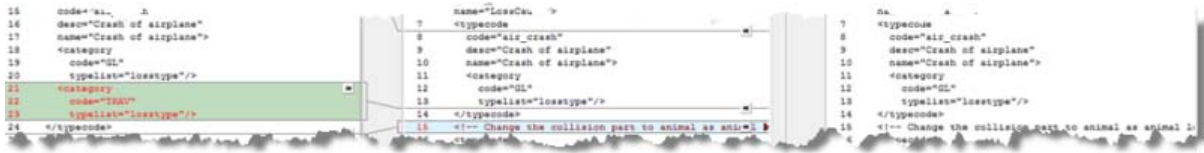
Referenced typecode status	Action
new: exists only in target version	Merge in the new typecode and merge in the referenced typecode in its typelist. If you do not want to use the new typecode, retire it by setting the <code>retired</code> attribute of the typecode to <code>true</code> .
active: exists in base or custom version and is not retired	Do not merge the new typecode.
retired: exists in base or custom version and is retired	Merge in the new typecode. If you do not want to use the new typecode, retire it by setting the <code>retired</code> attribute of the typecode to <code>true</code> .

In the following example, the typecode `abandonment` did not exist previously, either in the old default configuration or in the custom configuration. If you do not want the new typecode, Guidewire recommends that you merge and retire the typecode and all its children. This assumes that you are following other recommendations such that the `TRAV` losstype referenced is also merged in, and retired if applicable. Because this typecode is retired, there are no harmful consequences of this action. If you choose to unretire the typecode in the future, you will have the desired, default behavior in place.



In the next example, `air_crash` is a typecode that is already existing and active in the original base version. The latest version introduces a new `<category>` losstype subelement of `TRAV` to the `air_crash` typecode. If `TRAV` is already an active typecode, do not merge in the new `<category>` subelement. If you do, you will create a line of

business relationship in the data model which did not previously exist. Merging in category subelements that reference active typecodes might produce unwanted consequences.



Note: The LineCategory typelist tends to be very large. Guidewire recommends that you select the **Copy Prior Customized** option and not merge the LineCategory typelist. If you are keeping your existing line of business configuration, your customized LineCategory typelist is sufficient for the upgrade.

Merging Entities

Guidewire recommends that you merge in newly defined indexes from the target configuration.

For example, in the Claim.etx file, merge in the following changes to your customized configuration:

```
<index
  desc="Covering index for helping to speed up Claim Search by Insured's last name, when including
        lossdate as one of the criteria"
  expectedtobecovers="true"
  name="claimu6"
  trackUsage="true"
  unique="true">
  <indexcol
    keyposition="1"
    name="InsuredDenormID"/>
  <indexcol
    keyposition="2"
    name="Retired"/>
  <indexcol
    keyposition="3"
    name="LossDate"/>
  <indexcol
    keyposition="4"
    name="ID"/>
</index>
```

Merging Contact Entities That Have Foreign Key or Array Extensions

If you have extended the Contact entity or any Contact subentities by adding foreign keys or arrays, you must edit each added entity and have it implement AddressBookLinkable. Additionally, if you have extended ABContact or any of its subentities in ContactCenter in the same way, these added entities must each implement ABLinkable.

For example:

- If you extended the Contact entity in ClaimCenter with an array or foreign key, you would add the following element to the .etx file for the new entity. Add the element to the entity that populates the array or the entity to which the foreign key links:

```
<implementsEntity name="AddressBookLinkable"/>
```

- If you extended the ABContact entity in ContactCenter with an array or foreign key, you would add the following element to the .etx file for the new entity. Add the element to the entity that populates the array or the entity to which the foreign key links:

```
<implementsEntity name="ABLinkable"/>
```

Merging Other Files

In some cases, the differences between files cannot be merged effectively using a comparison tool. In particular, config.xml, logging.properties, and scheduler-config.xml often have many changes between major versions. Consider adding your custom changes to the new Guidewire-provided version instead of merging from prior versions if the presentation of these files in the merge tool is too daunting.

About Display Properties

The Configuration Upgrade Tool marks display properties files, such as `display.properties`, as unmergeable. You do not need to manually merge display properties files. ClaimCenter collects display properties from both the `cc` and `configuration` modules. This is different than the way ClaimCenter handles most other files. For most files, if a version exists in the `configuration` module, ClaimCenter uses that version instead of the version in the `cc` module. For display properties files, ClaimCenter creates a union of values from files in the `cc` and `configuration` modules each time it starts.

If you have added locales, you can export a full list of display keys and typelists from the default ClaimCenter 6.0.8 locale to any locale you have defined. This list includes a section for display keys and typelists that do not yet have values defined for your locale. You can use this list to determine which display keys and typelists require localized values. You can then specify those values and import the list.

See “Translating New Display Properties and Typecodes” on page 43.

Upgrading Rules to ClaimCenter 6.0.8

The Configuration Upgrade Tool does not upgrade rules. The tool classifies rules in the unmergeable filter. Within the target directory, Guidewire-provided default rules are located in `modules/cc/config/rules` and `modules/pl/config/rules`. The Configuration Upgrade Tool moves your custom rules to `modules/configuration/config/rules`.

Guidewire also copies the default rules for the current release into a ClaimCenter 6.0.8 folder within `modules/configuration/config/rules/rules`. Update your rules in Studio. You can use the default rules in the ClaimCenter 6.0.8 folder as a comparison. Compare your custom rules to the new default 6.0.8 versions and update your rules as needed.

You might find it useful to do a bulk comparison of default rules from the base release against the 6.0.8 versions to determine what types of changes Guidewire has made.

To compare rules between versions using the Rule Repository Report

1. If you want to compare default rules only, temporarily remove custom rules from your starting version by moving the `modules/configuration/config/rules` directory to a location outside the ClaimCenter directory.
If you want to compare your custom rules against the ClaimCenter 6.0.8 rules, do not move the `modules/configuration/config/rules` directory from your starting version. However, do remove the `ClaimCenter<base version>` directory from `modules/configuration/config/rules/rules` of the starting version if this directory exists.
2. Open a command window.
3. Navigate to the `bin` directory of your starting version.
4. Enter the following command:

```
gwcc regen-rulereport
```

This command creates a rule repository report XML file in `build/cc/rules`.
5. Append the starting version number to the XML file name.
6. Restore moved directories to the starting version.
7. Temporarily copy `modules/configuration/config/rules` from the target version to a location outside the ClaimCenter directory. Removing this directory prevents your custom rules from being listed in the report. Because the Configuration Upgrade Tool does not modify your custom rules, there is no reason to compare your custom rules in the target configuration.

8. Navigate to the `bin` directory of your target ClaimCenter 6.0.8 version.
9. Enter the following command:

```
gwcc regen-rulereport
```

This command creates a rule repository report XML file in `build/rules`. There is a slight change to the path between the versions.
10. Append the target version number to the XML file name.
11. Restore the `modules/configuration/config/rules` directory to the target version.
12. Open both rule report XML files in a merge tool. You do not merge base rules using the rule repository reports. However, looking at changes that Guidewire has made to the base rules can help you determine the types of changes you must make in your custom rules.

In your merge tool, disable whitespace differences and comments to reduce the amount of inconsequential differences shown between rules.

Update custom rules using Studio. Studio does not compare your rules directly with target rules. However, Studio provides powerful Gosu editing features not available in a standard text editor that can alert you to issues.

When you have finished updating all of your custom rules, delete the ClaimCenter 6.0.8 rules directory from `modules/configuration/config/rules/rules`.

The ClaimCenter 6.0.8 default rules are enabled because some features depend on these rules.

Rules Required for Key Features

Some rules provided with ClaimCenter must be active for certain functionality to work. These rules are listed below by the feature for which they are required. Review these rules and determine which of the applicable functionality you want in your implementation.

Contact Automatic Synchronization

CCL02000 - Suspend AutoSync for Related Contacts

COP01000 - Update Check Address

Catastrophe Bulk Association

CPU09000 - Related to Catastrophe

CPU13000 - Catastrophe History

CLV10000 - Catastrophe

Claim Health Metrics

CPU18000 - Update Claim Health

EPU07000 - Update Claim Health

TPU07000 - Update Claim Health

Deductible Handling

EPU04000 - Update Deductible On Updated Exposure Coverage

EPU05000 - Update Deductible On Updated Coverage Deductible

EPU06000 - Stop Closing Of Exposure With Unpaid Deductible

TPU06000 - Unlink Deductible After Check Denial

Reinsurance

TPU04000 - Reinsurance

ISO Validation

CLV09000 - ISO Validation

Translating New Display Properties and Typecodes

ClaimCenter 6.0.8 adds new display properties and typecodes. If you have defined additional locales, export these new display properties and typecodes to a file, define localized values, and reimport the localized values. If you do not have additional locales defined in your ClaimCenter environment, skip this procedure.

To localize new display properties and typecodes

1. Open Studio from your ClaimCenter 6.0.8 environment by entering the following command from `ClaimCenter\bin`:

```
gwcc studio
```
2. In the Studio Resources pane, right-click **configuration** → **Typelists**.
3. Select **Export translation file** → **English (US) to locale**, where *locale* is the name of the locale for which you want to translate the new display properties and typecodes. A file browser window opens.
4. Choose a location and name for the exported file.
5. Open the file in a text editor. The first section of the file lists display properties and typecodes that have a localized value. The second section lists display properties and typecodes that do not have a localized value.
6. Specify localized values for the untranslated properties.
7. Save the updated file.
8. In the Studio Resources pane, right-click **configuration** → **Typelists**.
9. Select **Import translation file** → **locale**, where *locale* is the name of the locale for which you want to import the localized display properties and typecodes. A file browser window opens.
10. Use the file browser window to navigate to the file containing translated display properties and typecodes.
11. Click **Open**. Studio imports the localized typecodes and display keys. Studio also copies localized display keys to a `modules/configuration/config/locale/locale/display.properties` file.

After you import the localized typecodes and display keys, you can view them in Studio.

Modifying PCF files, Rules and Libraries for Unused Contact Subtypes

You might not want to use all of the contact subtypes provided with ClaimCenter. However, contact subtypes are referenced within PCF files (particularly in modal pages of the address book tab and ContactCenter), rules, and libraries. Therefore, Guidewire recommends that you do not remove the unused contact subtypes. Instead, modify PCF files, rules and libraries that reference either the specific unused subtypes or the `ContactSubtype` entity itself.

To find contact subtype references

1. Open Guidewire Studio using the `gwcc studio` command in `ClaimCenter/bin`.

2. Right-click **Rule Sets, Classes, or Page Configuration (PCF)** and select **Find in Path**.
3. For **Text to find**, enter `ContactSubtype`.
4. Click **Find**.
5. Repeat this procedure, searching for unused contact subtype names instead of `ContactSubtype`. For example, if you are not using the `Adjudicator` contact subtype, search for `Adjudicator` to see which files, rules and libraries reference this unused subtype.

After you have found all references to the `ContactSubtype` entity and the specific unused contact subtypes, review each case to determine how to proceed.

If the usage is in a range input, as in `AddressBookSearchDV.pcf`, use a filter to exclude the unused contact subtypes from the range input. Alternatively, you can set the filter to only include the contact subtypes that you want to use.

If a menu item uses the contact subtype, as in `AddressBookMenuActions.pcf`, remove the menu item to prevent users from performing actions with the unused contact subtype.

Validating the ClaimCenter 6.0.8 Configuration

This topic includes procedures to validate the upgraded configuration.

Using Studio to Validate Files

You can use Studio to validate files and rules without having to start ClaimCenter. Do not start ClaimCenter at this point. Studio can run without connecting to the application server.

To fix all problems in classes and enhancements including libraries, as well as PCF files and typelists:

1. Run the **Validate** option in the **Studio Tools** menu. This identifies all incorrect Gosu syntax in all Studio resources, issuing either a warning or an error.
2. Correct all identified errors with Studio. You can defer fixing warnings.

Starting ClaimCenter and Resolving Errors

IMPORTANT In the process described in this section, do not point the ClaimCenter server at a production database. The goal of this process is to test the configuration upgrade. Create an empty database account and point ClaimCenter to this account for this process. See “Configuring the Database” on page 20 in the *Installation Guide* and “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Upon starting the server for the first time, you might receive errors that prevent the server from starting. In general, fixing errors and starting the server is an iterative process that involves:

1. Start the server for the target configuration.
ClaimCenter encounters a configuration error and shuts down.
2. Copy the error message to a log file.
3. Locate the configuration causing the error - for example, a line of code in a PCF.
4. Comment out the offending line.

After the server starts successfully, look at the log and start solving errors and introducing solutions into the environment. Assign errors to developers on your team.

5. Copy the commented file to the test bed for later analysis.
6. Begin again with step 1. Continue until the server starts successfully.

When the server starts successfully, resolve any remaining issues in the configuration that caused startup errors. Attempt to resolve each error individually and start the server to see if the fix worked.

Building and Deploying ClaimCenter 6.0.8

After you apply and validate an upgrade to the configuration environment, rebuild and redeploy ClaimCenter. Before you begin, make sure you have carefully prepared for this step. In particular, make sure you have updated your infrastructure appropriately.

Review this topic and then rebuild and redeploy ClaimCenter to the application server. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide* of the target version for instructions.

WARNING Do not yet start ClaimCenter. Only package the application file and deploy it to the application server. Starting ClaimCenter begins the database upgrade.

If you have multiple Guidewire products to upgrade, such as ContactCenter, upgrade, build, and deploy each individually before attempting to re-integrate them.

The Build Environment

With the exception of the database configuration, the first time you start the application server use the unmodified `config.xml` and `logging.properties` files provided with the target configuration. After the server starts successfully, you can merge in specific configurations of these files.

If you are using a `build.xml` file to run the `ClaimCenter/bin` build tasks, ensure that it is updated correctly for the target infrastructure. You might encounter problems running build scripts if the data dictionary generation fails due to a PCF validation error. However, this dictionary does not report these errors.

If you encounter build failures due to data dictionary generation, you can comment out this dictionary generation. Then, as you start the server, it reports any PCF configuration errors. After you have corrected PCF configurations, un-comment the dictionary generation and rebuild the application file.

Preserving JAR Files

Place custom JAR files in the `/config/lib` directory. The `build-war`, `build-weblogic-ear`, or `build-websphere-ear` command and subsequent deployment of the packaged application copies the JAR file into the appropriate place for it to be accessed by ClaimCenter. JAR files in this location survive the upgrade process.

Upgrading the ClaimCenter 6.0.x Database

This topic provides instructions for upgrading the ClaimCenter database to ClaimCenter 6.0.8.

This topic includes:

- “Identifying Data Model Issues” on page 48
- “Verifying Batch Process and Work Queue Completion” on page 49
- “Purging Data Prior to Upgrade” on page 49
- “Validating the Database Schema” on page 50
- “Checking Database Consistency” on page 51
- “Creating a Data Distribution Report” on page 51
- “Generating Database Statistics” on page 52
- “Creating a Database Backup” on page 53
- “Updating Database Infrastructure” on page 53
- “Preparing the Database for Upgrade” on page 53
- “Handling Extensions” on page 54
- “Setting Linguistic Search Collation for Oracle” on page 55
- “Using the IDatabaseUpgrade Plugin” on page 56
- “Disabling the Scheduler” on page 64
- “Suspending Message Destinations” on page 64
- “Configuring the Database Upgrade” on page 65
- “Starting the Server to Begin Automatic Database Upgrade” on page 68
- “Viewing Detailed Database Upgrade Information” on page 72
- “Dropping Unused Columns on Oracle” on page 73

- “Setting Claim Descriptions” on page 73
- “Final Steps After The Database Upgrade is Complete” on page 74

Identifying Data Model Issues

Before you upgrade a production database, identify issues with the datamodel by running the database upgrade on an empty database. This process does not identify all possible issues. The database upgrade does not detect issues caused by specific data in your production database. Instead, this procedure identifies issues with the data model.

Complete the following procedure to identify data model issues, and correct any issues on an empty schema. Then, follow the full list of procedures in this topic to upgrade a production database. This list begins with “Verifying Batch Process and Work Queue Completion” on page 49 and finishes with “Final Steps After The Database Upgrade is Complete” on page 74.

To identify data model issues

1. Create an empty schema of your starting version database. You can do this in a development environment by pointing the development ClaimCenter installation at an empty schema and starting the ClaimCenter server. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.
2. Complete the configuration upgrade for data model files in your starting version, according to the instructions in the previous topic. You do not need to complete the merge process for all files. See also “Handling Extensions” on page 54.
3. Configure your upgraded development environment to point to the database account containing the empty schema of your old version. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.
4. Start the ClaimCenter server in your upgraded development environment. The server performs the database upgrade to ClaimCenter 6.0.8. See “Starting the Server to Begin Automatic Database Upgrade” on page 68.
5. Check for errors reported during the upgrade process. Resolve any issues before upgrading your production database.

These errors can be issues detected by the database upgrade version checks or issues with the domain and admin graphs. ClaimCenter uses the domain and admin graphs for a number of features including archiving and purging. ClaimCenter 6.0.8 performs a number of graph validation checks when you start the server. The ClaimCenter server does not start if certain graph validation checks fail.

See “Graph Validation Checks” on page 64 in the *System Administration Guide* for a description of the graph validation checks that ClaimCenter performs.

See “The Domain Graph” on page 279 in the *Configuration Guide* for a description of the domain and admin graphs. That topic includes information about configuring the data model properly to maintain graphs suitable for ClaimCenter.

You can use the IDatabaseUpgrade plugin to run custom SQL before and after the database upgrade. For more information, see “Running Custom SQL” on page 57.

If ClaimCenter detects circular references when the server starts, you can use IDatabaseUpgrade plugin to convert foreign keys to edge foreign keys during the major version database upgrade. See “Eliminating Circular References” on page 59.

Verifying Batch Process and Work Queue Completion

All batch processes and distributed work queues must complete before beginning the upgrade. Check the status of batch processes and work queues in your current production environment.

To check the status of batch processes and work queues

1. Log in to ClaimCenter as the superuser.
2. Press **Alt + Shift + T**. ClaimCenter displays the **Server Tools** tab.
3. Click **Batch Process Info**.
4. Select **Any** from the **Processes** drop-down filter.
5. Click **Refresh**.
6. Check the **Status** column for each batch process listed. This list also includes batch processes that are writers for distributed work queues. If any of the batch processes have a **Status** of **Active**, wait for the batch process to complete before continuing with the upgrade.

Purging Data Prior to Upgrade

This topic includes recommendations for purging certain types of data from the database prior to upgrade. Removing unused records can improve the performance of the database upgrade and ClaimCenter.

Purging Old Messages from the Database

Purge completed inactive messages before upgrading the database. Doing so reduces the complexity of the database upgrade.

Use the following command from the current (pre-upgrade) customer configuration `admin/bin` directory to purge completed messages from the `cc_MessageHistory` table:

```
messaging_tools -password password -server http://server:port/instance -purge MM/DD/YYYY
```

This tool deletes completed messages with a send time before the date *MM/DD/YYYY*.

Or, you can use the following web service API:

```
IMessageToolsAPI.purgeCompletedMessages(java.util.Calendar cutoff)
```

Or, you can use the **Purge Message History** batch process. 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.

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

```
maintenance_tools -password password -startprocess PurgeMessageHistory
```

Periodically purge old messages to prevent the database from growing unnecessarily.

Purge messages from the database before starting ClaimCenter, so the database upgrade does not attempt to convert those rows.

You cannot resend old messages after the upgrade. This is because integrations change and the message payload might be different. It is important that messages that have failed or not yet been consumed finish prior to upgrading.

After you purge completed inactive messages, reorganize the `cc_MessageHistory` table. You might also want to rebuild any indexes on the table. Contact Guidewire Support if you need assistance.

Purging Orphaned Policies from the Database

Each claim has an associated policy record. In ClaimCenter 6.0 and prior versions, if a user refreshed the policy information on a claim, the old policy record remained in the database. If orphaned policies are consuming large amounts of database space, contact Guidewire Support for assistance removing the policies.

Purging Address Correction Records

The `cc_AddressCorrection` table stores address corrections returned from geocoding systems. ClaimCenter does not display address corrections by default. However, you might have configured ClaimCenter to expose address corrections to allow users to make corrections.

If you have not configured ClaimCenter to expose address corrections, you can remove the address correction records by truncating the `cc_AddressCorrection` table.

If you do have address corrections exposed, you can remove records that have been handled already.

Purging Completed 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
```

Validating the Database Schema

This validation detects the unlikely event that the data model defined by your configuration files has become out of sync with the database schema. While the pre-upgrade server is running, use the `system_tools` command in `admin/bin` of the customer configuration to verify the database schema:

```
system_tools -password password -verifydbschema -server servername:port/instance
```

Correct any validation problems in the database before proceeding. Contact Guidewire Support for assistance.

Following the database upgrade, run this command again from the admin/bin directory of the target (upgraded) configuration.

Checking Database Consistency

ClaimCenter has hundreds of internal database consistency checks. Before upgrading, run consistency checks by executing the following command from the admin/bin directory of the customer configuration.

```
system_tools -password password -checkdbconsistency -server servername:port/instance
```

This command takes a long time and could time out. If it does, run the command on subsets of the database instead of the entire database. See the “system_tools Command” on page 173 in the *System Administration Guide* for instructions.

You can also trigger database consistency checks to run at server startup by setting `checker=true` in the database block of `config.xml`.

```
<database name="ClaimCenterDatabase" driver="dbcp"
  dbtype="sqlserver" autoupgrade="false" checker="true">
  ...
</database>
```

Each time the server starts with `checker=true`, it will run database consistency checks. These checks might take a long time to run. The server is not available until the consistency checks are completed. Therefore, when you have finished, disable consistency checks for the next server startup by setting `checker=false`.

Run database consistency checks early in the upgrade project. Fix any consistency errors. Continue to periodically run consistency checks and resolve any issues so that your database is ready to upgrade when you begin the upgrade procedure. Consistency issues might take some time to resolve, so begin the process of running consistency checks and fixing issues early.

IMPORTANT Resolve consistency check errors prior to upgrade, especially those surrounding Financials and Incident creation. After resolving any data issues, run the consistency checks again to ensure the database is ready to be upgraded. Contact Guidewire Support for information on how to resolve any consistency issues.

Following the database upgrade, run this command again from the admin/bin directory of the target (upgraded) configuration.

Creating a Data Distribution Report

Generate a data distribution report for the database before an upgrade. Save the output of this report. Run the report again after the upgrade to ensure the distribution is still correct.

Guidewire is very interested in the data distribution of your databases. Guidewire uses these reports to better understand the nature of your database and to optimize ClaimCenter performance. Guidewire appreciates copies of your reports, both before and after upgrades.

You can also use this information to tune the application server cache. See “Understanding Application Server Caching” on page 70 in the *System Administration Guide*.

To create a database distribution report

1. In `config.xml`, set `<param name="EnableInternalDebugTools" value="true"/>`.

2. Start the ClaimCenter application server.
3. Log into ClaimCenter as an administrative user.
4. Type ALT + SHIFT + T while in any screen to reach the **Server Tools** page.
5. Choose **Info Pages** from the **Server Tools** tab.
6. Choose the **Data Distribution** page from the **Info Pages** dropdown.
7. Enter a reason for running the Data Distribution batch job in the **Description** field.
8. On this page, select the **Collect distributions for all tables** radio button and check all checkboxes to collect all distributions.
9. Push the **Submit Data Distribution Batch Job** button on this page to start the data collection.
10. Return to the **Data Distribution** page and push its **Refresh** button to see a list of all available reports. The batch job has completed when the **Available Data Distribution** list on the **Data Distribution** page includes your description.
11. Select the desired report and use the **Download** button to save it zipped to a text file. Unzip the file to view it.

Generating Database Statistics

To optimize the performance of the ClaimCenter database, it is a good idea to update database statistics on a regular basis. Both SQL Server and Oracle can use these statistics to optimize database queries. Before you upgrade, and before you go into production, update the database statistics by running the `maintenance_tools` command from `admin/bin` of the customer (pre-upgrade) configuration.

To generate database statistics

1. Run the `maintenance_tools` command to get the proper SQL statements for updating the statistics in ClaimCenter tables:

```
maintenance_tools -getdbstatisticsstatements -password password  
-server http://server:port/instance> db_stats.sql
```

2. Run the resulting SQL statements against the ClaimCenter database.
3. If using an Oracle database, the `maintenance_tools` command creates `db_stats.ora`. Review and then run these statements against the database.

You can configure a SQL Server instance to periodically update statistics using SQL. See your database documentation and “Configuring Database Statistics” on page 53 in the *System Administration Guide* for more information.

The database upgrade can take a long time, and has built-in statistics collection that help you see if any part of the upgrade is slow. Collect these statistics, and compare them to the statistics you collected before the upgrade. The `config.xml` file has parameters that control this statistics collection.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Creating a Database Backup

Prepare the environment so that you can make a total recovery of the original installation if you run into problems during the upgrade.

The first time you start the ClaimCenter server after running the upgrade tool, the server updates the database. During its work, the database upgrader minimizes the logging that it does. For these reasons, back up your database before starting an upgrade. Your pre-upgrade database might not be recoverable after an upgrade.

Updating Database Infrastructure

Before starting the upgrade, update database server software and operating systems as needed to meet the installation requirements of ClaimCenter 6.0.8. 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>.

Preparing the Database for Upgrade

This topic notes steps to prepare the database for the upgrade process.

Ensuring Adequate Free Space

The database upgrade requires significant free space. Make sure the database has at least 50% of the current database size available as free space.

Disabling Replication

Disable database replication during the database upgrade.

Assigning Default Tablespace (Oracle only)

Set the default tablespace for the database user to the one mapped to the logical tablespace OP in `config.xml`.

The database upgrade creates temporary tables during the upgrade without specifying the tablespace. If the Oracle database user was created without a default tablespace, Oracle by default creates the tables in the SYSTEM tablespace. The Guidewire database user is likely not to have the required quota permission on the SYSTEM tablespace. This results in an error of the type:

```
java.sql.SQLException: ORA-01950: no privileges on tablespace 'SYSTEM'
```

Even if the default tablespace is not SYSTEM, if the Guidewire database user does not have quota permission on the default tablespace, the temporary table creation during upgrade fails.

Using Proper Clock Settings

The `config.xml` file contains several parameters set to AM/PM times (for example, `<param name="BusinessDayEnd" value="5:00 PM"/>`). If your server uses a 24-hour clock, change these parameters to reflect the server clock (`<param name="BusinessDayEnd" value="17:00"/>`).

Handling Extensions

This topic discusses how to handle extensions during the upgrade to ClaimCenter 6.0.8.

Merging Extensions

ClaimCenter 6.0.8 stores extensions in `.eti` and `.etx` files. An *Entity.eti* file defines a new entity. An *Entity.etx* file defines extensions to an existing entity. The Configuration Upgrade Tool compares these files against the extension files included with ClaimCenter 6.0.8.

Guidewire often adds indexes to entities in the target configuration to improve the performance of database queries in ClaimCenter 6.0.8. ClaimCenter requires some of these indexes. Guidewire adds required indexes to entity definitions in the data model. Other indexes are recommended for most installations but can be disabled if they negatively impact performance. Guidewire adds optional indexes to entity extensions so you can disable any if necessary.

Use the Configuration Upgrade Tool to resolve extensions files. When you merge your custom extensions with Guidewire changes, review each new index added by Guidewire. In most cases, include the new index in the merged extension file. You can modify or remove index definitions based on usage in your deployment.

Reviewing Custom Extensions

Generate and review the data dictionary for the target version to identify any custom extensions that are now obsolete due to Guidewire adding a similar field to the base ClaimCenter.

To generate the data dictionary

1. From the command line, navigate to the `bin` directory of the target version.
2. Run the command `gwcc regen-dictionary`.

This command generates the data and security dictionaries in the `build/dictionary` directory of the target version. To view the data dictionary, open `build/dictionary/data/index.html` in a web browser.

Compare the target version data dictionary with the version in your current environment. If you have extensions that are now available as base fields, consider migrating the data in those fields to the base version. Consider whether an extension is still on the appropriate entity. A new entity could be a more appropriate location for the extension. Review key data model changes that might impact your custom extensions.

If you change an extension location or migrate to a new base field, update any PCF, rule or library that references the extension to reference the new location.

Reconciling the Database with Custom Extensions

The extensions defined in `.eti` and `.etx` files must match the physical database. Delete all physical columns in the database that are not part of the base installation or defined as extensions before starting the server.

Setting Linguistic Search Collation for Oracle

IMPORTANT This section applies for Oracle implementations only. On SQL Server, ClaimCenter uses the collation setting of the database server. Database searching on SQL Server is always case-insensitive, and obeys the rules of the Windows collation set on the database. This section does not apply for SQL Server implementations.

WARNING Oracle Java Virtual Machine (JVM) must be installed on all Oracle databases hosting ClaimCenter. The only exception is when the ClaimCenter application locale is English and you only require case-insensitive searches. Ensure that Oracle initialization parameter `java_pool_size` is set to a value of above 50 MB.

You can specify how you want ClaimCenter to collate search results. The `strength` attribute of the `LinguisticSearchCollation` element of `GWLocale` for the default locale in `localization.xml` specifies how ClaimCenter sorts search results. You can set the strength to `primary` or `secondary`.

With `LinguisticSearchCollation strength` set to `primary`, ClaimCenter searches results in a case-insensitive and accent-insensitive manner. ClaimCenter considers an accented character equal to the unaccented version of the character if the `LinguisticSearchStrength` for the default application locale is set to `primary`. For example, with `LinguisticSearchCollation strength` set to `primary`, ClaimCenter treats “Reneé”, “Renee”, “renee” and “reneé” the same.

With `LinguisticSearchCollation strength` set to `secondary`, ClaimCenter searches results in a case-insensitive, accent-sensitive manner. ClaimCenter does not consider an accented character equal to the unaccented version of the character if the `LinguisticSearchCollation strength` for the default application locale is set to `secondary`. For example, with `LinguisticSearchCollation strength` set to `secondary`, a ClaimCenter search treats “Renee” and “renee” the same but treats “Reneé” and “reneé” differently. By default, ClaimCenter uses a `LinguisticSearchCollation strength` of `secondary`, for case-insensitive, accent-sensitive searching.

For Oracle, the following rules apply:

- **Case:** All searches ignore the case of the letters, whether `LinguisticSearchCollation strength` is set to `primary` or `secondary`. “McGrath” equals “mcgrath”.
- **Punctuation:** Punctuation is always respected, and never ignored. “O'Reilly” does not equal “OReilly”.
- **Spaces:** Spaces are respected. “Hui Ping” does not equal “HuiPing”.
- **Accents:** An accented character is considered equal to the unaccented version of the character if `LinguisticSearchCollation strength` is set to `primary`. An accented character is not equal to the unaccented version if `LinguisticSearchCollation strength` is set to `secondary`.

Japanese only

- **Half Width/Full Width:** Searches under a Japanese locale always ignore this difference.
- **Small/Large Kana:** Japanese small/large letter differences are ignored only when `LinguisticSearchCollation strength` is set to `primary`, meaning accent-insensitive.
- **Katakana/Hiragana sensitivity:** Searches under a Japanese locale always ignore this difference.
- The long dash character is always ignored.
- Soundmarks (`` and °) are only ignored if `LinguisticSearchCollation strength` is set to `primary`.

German only

- Vowels with an umlaut compare equally to the same vowel followed by the letter e. Explicitly, “ä”, “ö”, “ü” are treated as equal to “ae”, “oe” and “ue”.
- The Eszett, or sharp-s, character “ß” is treated as equal to “ss”.

ClaimCenter populates denormalized values of searchable columns to support the search collation. For example, with `LinguisticSearchCollation` strength set to `primary`, ClaimCenter stores the value “Reneé”, “Renee”, “renee” and “renee” in a denormalized column as “renee”. With `LinguisticSearchCollation` strength set to `secondary`, ClaimCenter stores a denormalized value of “renee” for “Renee” or “renee” and stores “renee” for “Reneé” or “renee”. Japanese and German locales make additional changes when storing values in denormalized columns in order to conform to the rules listed previously for those locales.

Any time you change the `LinguisticSearchCollation` strength and restart the server, ClaimCenter repopulates the denormalized columns. Previous versions of ClaimCenter populated the denormalized columns with lowercase values for case-insensitive search, equivalent to setting `LinguisticSearchCollation` strength to `secondary`. If you set `LinguisticSearchCollation` strength to `primary`, ClaimCenter repopulates the denormalized columns, substituting any accented characters for their base equivalents. This process can take a long time, depending on the amount of data. Therefore, if you want to change `LinguisticSearchCollation` strength to `primary`, you might want to do so after the database upgrade. If you are concerned about the duration of the database upgrade, you can change your search collation settings after the upgrade. During a maintenance period, change `LinguisticSearchCollation` strength to `primary` and restart the server to repopulate the denormalized columns.

For Japanese locales, the ClaimCenter database upgrade from a prior major version repopulates the denormalized columns regardless of the `LinguisticSearchCollation` strength value. ClaimCenter must repopulate the denormalized columns for Japanese locales to have search results obey the Japanese-only rules listed previously.

Using the IDatabaseUpgrade Plugin

The `IDatabaseUpgrade` plugin interface provides hooks for custom code that you want to run during the database upgrade. You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. You can also use the `IDatabaseUpgrade` plugin to eliminate circular references in your data model by converting specific foreign keys to edge foreign keys.

WARNING The `IDatabaseUpgrade` plugin runs any time that the server determines that a database upgrade is necessary, whether for a major or minor version. Therefore, use conditionals within your `preUpgrade` and `postUpgrade` methods and `get EdgeForeignKeyUpgrades` property to control execution of your custom code. Test these conditionals thoroughly to ensure that the custom code you define within the methods runs appropriately. Remove or disable the `IDatabaseUpgrade` plugin once you have completed the upgrade.

The `IDatabaseUpgrade` plugin interface includes the following properties:

```
get BeforeUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
get AfterUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
```

These properties are for making custom upgrades to archived XML entities. This is only applicable to implementations that have enabled archiving. If you do not have archiving enabled, return an empty list for these properties.

```
override property get BeforeUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}

override property get AfterUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
```



```

        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();
    return list;
}

```

If you make custom data model changes within ClaimCenter and enable archiving, you can use these properties to define changes for the upgrader to make to the archived XML. See “Upgrading Archived Entities” on page 62.

Running Custom SQL

You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. For example, you might fix a consistency check failure issue, correct issues reported by version checks, or delete a custom extension that you are no longer using.

The `IDatabaseUpgrade` plugin interface contains method signatures for two methods that you must define in your plugin. These signatures are:

- `function preUpgrade(context : IUpgradeContext)`
- `function postUpgrade(context : IUpgradeContext)`

The `preUpgrade` method runs before version checks, so you can include any SQL to resolve version check issues in the `preUpgrade` method. You can also write SQL to resolve consistency check issues among other operations.

The `postUpgrade` method runs after the database upgrade version triggers. You can incorporate SQL in the `postUpgrade` method to move data into tables or columns that did not exist prior to upgrading.

The `IDatabaseUpgrade` interface also contains an `Iterable` property used for eliminating circular references. You must define this property. If you are not using this feature of the `IDatabaseUpgrade` plugin, you can define an empty `Iterable` as follows:

```

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
        return {}
    }

```

Any custom SQL that you write must be idempotent. That is, the SQL must be designed so that it can run against the database multiple times without corrupting data. The `IDatabaseUpgrade` plugin runs each time the database is upgraded, whether for a major or minor version of ClaimCenter or for a data model change such as a new extension. Check the version of the database in your `preUpgrade`, `postUpgrade` methods, and `get EdgeForeignKeyUpgrades` property to ensure that your custom SQL is executed appropriately. Guidewire recommends that you create a custom method to check the database version.

For ClaimCenter 5.0 you can determine the database major and minor version by opening `modules\cc\config\metadata\metadataproperties.xml` and checking the value of the `majorVersion` and `minorVersion` properties.

For 6.0 versions of ClaimCenter, the internal major and minor version numbers are listed in `modules\cc\config\metadata\metadata.properties`.

You can also determine the major and minor version of a ClaimCenter instance by reading the messages reported by the server during startup.

```

[java] serverName timestamp INFO Validating main database connections have all required properties
[java] serverName timestamp INFO Checking if existing database version is current...
[java] serverName timestamp INFO Datamodel version (major, minor, platform) is up to date.

```

This message reports the internal major, minor and platform version numbers. Only the major and minor version are of interest for writing custom SQL. Note that these are internal version numbers and do not match the commonly used external version number, such as 6.0.8.

To run custom SQL

1. Create a new Gosu class that implements `IDatabaseUpgrade`. If you already created a Gosu class to eliminate circular references, you can use the `preUpgrade` and `postUpgrade` methods of that class. The class you create must define the `preUpgrade` and `postUpgrade` methods and the `get EdgeForeignKeyUpgrades` property.

The following example demonstrates using the `preUpgrade` method to set `Claim.DeductibleStatus`, `Claim.ReinsuranceStatus` and `Claim.LossLocationID` to null:

```
package gw.myplugins.upgrade.impl

uses gw.plugin.upgrade.IDatabaseUpgrade
uses gw.plugin.upgrade.IUpgradeContext
uses java.lang.Iterable
uses gw.plugin.upgrade.EdgeForeignKeyUpgradeInfo
uses gw.api.archiving.upgrade.ArchivedDocumentUpgradeVersionTrigger

class DatabaseUpgradeImpl implements IDatabaseUpgrade {

    construct() {
    }

    // custom function to determine if DB version is prior to ClaimCenter 6.0 (DB version 8).
    // MajorVersion of CC 5.x is 7, MajorVersion for CC 6.x is 8

    private function isMajorVersionLessThan8(upgradeContext : IUpgradeContext) : boolean {

        //if null upgrade context is sent, return false and log reason
        if (upgradeContext == null) {
            //printing instead of logging because logs were not shown in console
            print("databaseupgrade.isMajorVersionLessThan8() - null upgradeContext sent")
            return false
        }

        //try to access major version, was throwing exception
        try {

            if (upgradeContext.CurrentMajorVersion < 8) {
                return true
            }

        } catch(e) {

            //swallow exception and return false
            print("databaseupgrade.isMajorVersionLessThan8() - accessing " +
                "upgradeContext.CurrentMajorVersion threw exception. Swallowing following exception" +
                "and returning false.")

            e.printStackTrace()

            return false
        }

        //All other conditions, return false
        return false
    }

    override function preUpgrade(upgradeContext : IUpgradeContext) {

        // first check the major version of the database to determine if scripts need to be run.
        if (isMajorVersionLessThan8(upgradeContext)) {

            // DB will be upgraded to CC6.0, execute pre-upgrade scripts
            upgradeContext.update("Set DeductibleStatus and ReinsuranceStatus to null",
                "UPDATE CC_CLAIM " +
                "SET DeductibleStatus = null, ReinsuranceStatus = null " +
                "WHERE DeductibleStatus is not null OR ReinsuranceStatus is not null")

            upgradeContext.update("Null out LossLocationID",
                "UPDATE CC_CLAIM " +
                "SET LossLocationID = null " +
                "WHERE LossLocationID is not null")

        }
    }

    override function postUpgrade(upgradeContext : IUpgradeContext) {

        // first check the major version of the database to determine if scripts need to be run.
        if (isMajorVersionLessThan8(upgradeContext)) {

            ///# todo: Implement

        }
    }

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
```

```

        // If not applicable, return empty iterable.
        return {};
    }

    override property get BeforeUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }

    override property get AfterUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }
}

```

2. Implement the `IDatabaseUpgrade` plugin with the new class. If you already implemented this class to eliminate circular references, skip this step.

- a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
- b. In the Studio **Resources** pane, expand **Plugins** → **gw** → **plugin** → **upgrade**.
- c. Right-click **IDatabaseUpgrade** and select **Implement**.
- d. On the **IDatabaseUpgrade** tab, click **Add...**
- e. Select **Gosu**.
- f. Enter your class, including the package.
- g. Select **File** → **Save Changes**.

When you start the server to perform the database upgrade from a prior major version, the upgrade calls the plugin and runs your custom `preUpgrade` and `postUpgrade` methods.

Eliminating Circular References

ClaimCenter 6.0.5 and newer enforces data model integrity more strictly than previous versions. ClaimCenter runs a series of checks on the data model during server startup. If any of these checks fails, ClaimCenter reports an error and does not start. One of these checks looks for circular references in the data model.

A circular reference exists when an entity points to an entity of the same type through one or more foreign keys. The circular reference can be simple, in which the entity has a foreign key directly to the same entity. Or, the reference can be more complex, in which an entity has a foreign key to another entity that directly or indirectly references the original entity.

To resolve circular references in the data model, use the `IDatabaseUpgrade` plugin to convert foreign keys causing circular references to edge foreign keys. This plugin enables you to specify the entity, existing foreign key and the edge foreign key to create. The plugin is called during the database upgrade from a prior major version.

The interface also contains an `Iterable` property used for eliminating circular references. You must define this property. If you are not using this feature of the `IDatabaseUpgrade` plugin, you can define an empty `Iterable` as follows:

```

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
        return {}
    }
}

```

The `IDatabaseUpgrade` plugin interface also contains method signatures for methods that you must define in your plugin. These signatures are:

- `function preUpgrade(context : IUpgradeContext)`
- `function postUpgrade(context : IUpgradeContext)`

If you do not want to run custom SQL before or after the upgrade, you can create these methods with no operations defined within them.

To convert foreign keys to edge foreign keys

1. Change the data model to replace the foreign key with an edge foreign key.
 - a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
 - b. In the Studio **Resources** pane, expand **Data Model Extensions** → **extensions**.
 - c. Double-click *EntityName.etx* to open the file defining the entity extension.
 - d. Change the `foreignKey` element to `edgeForeignKey`.
 - e. Add an `edgeTableName` attribute and set the value to a name for the edge table, such as `edgeTableName="entitynameedge"`.
 - f. Remove the `columnName` attribute.
 - g. If you want the contents of the edge table to be part of the domain graph, add the following subelement to `edgeForeignKey`:

```
<implementsEntity name="Extractable"/>
```

 Be sure to move the closing tag of the `edgeForeignKey` element if necessary to encapsulate the subelement.
 - h. Select **File** → **Save Changes**.
 - i. Repeat steps c through h for each foreign key that you want to convert to an edge foreign key.
2. Create a new Gosu class that implements `IDatabaseUpgrade`. If you already created a Gosu class to run custom SQL before or after the upgrade, you can use the `get EdgeForeignKeyUpgrades` property of that class. The class you create must define the `get EdgeForeignKeyUpgrades` property and the `preUpgrade` and `postUpgrade` methods. If you do not want to run custom SQL before or after the upgrade, you can create these methods with no operations defined within them. For example:

```
package gw.myplugins.upgrade.impl
uses gw.plugin.upgrade.IDatabaseUpgrade
uses gw.plugin.upgrade.IUpgradeContext
uses java.lang.Iterable
uses gw.plugin.upgrade.EdgeForeignKeyUpgradeInfo
uses gw.api.archiving.upgrade.ArchivedDocumentUpgradeVersionTrigger

class DatabaseUpgradeImpl implements IDatabaseUpgrade {

    construct() {
    }

    private var _upgradeContext : IUpgradeContext

    // custom function to determine if DB version is prior to ClaimCenter 6.0 (DB version 8).
    // MajorVersion of CC 5.x is 7, MajorVersion for CC 6.x is 8

    private function isMajorVersionLessThan8(upgradeContext : IUpgradeContext) : boolean {
        _upgradeContext = upgradeContext

        //if null upgrade context is sent, return false and log reason
        if (upgradeContext == null) {
            //printing instead of logging because logs were not shown in console
            print("databaseupgrade.isMajorVersionLessThan8() - null upgradeContext sent")
            return false
        }

        //try to access major version, was throwing exception
    }
```

```

try {
    if (upgradeContext.CurrentMajorVersion < 8) {
        return true
    }
} catch(e) {

    //swallow exception and return false
    print("databaseupgrade.isMajorVersionLessThan8() - accessing" +
        "upgradeContext.CurrentMajorVersion threw exception. Swallowing following exception" +
        "and returning false.")

    e.printStackTrace()

    return false
}

//All other conditions, return false
return false
}

override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
    // Check the major version of the database to determine which edge foreign keys must be converted.
    // MajorVersion of CC5.x is 7, MajorVersion for CC6.x is 8
    if (isMajorVersionLessThan8(_upgradeContext)) {

        return {
            // add these as necessary
            // new EdgeForeignKeyUpgradeInfo("foreignKeyColumn", entityName, "edgeForeignKey")
        }
    }
    else {
        return {}
    }
}

override function preUpgrade(p0 : IUpgradeContext) {
}

override function postUpgrade(p0 : IUpgradeContext) {
}

override property get BeforeUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}

override property get AfterUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}
}

```

Add a new `EdgeForeignKeyUpgradeInfo("foreignKeyColumn", entityName, "edgeForeignKey")` line to the `get EdgeForeignKeyUpgrades()` definition for each foreign key that you want to convert. Separate these lines with commas.

3. Implement the `IDatabaseUpgrade` plugin with the new class. If you already implemented this class to run custom SQL before or after the database upgrade, skip this step.
 - a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
 - b. In the Studio Resources pane, expand **Plugins** → **gw** → **plugin** → **upgrade**.
 - c. Right-click `IDatabaseUpgrade` and select **Implement**.
 - d. On the `IDatabaseUpgrade` tab, click **Add...**
 - e. Select **Gosu**.

f. Enter your class, including the package.

g. Select **File** → **Save Changes**.

When you start the server to perform the database upgrade from a prior major version, the upgrade calls the plugin and converts the foreign keys to edge foreign keys.

Upgrading Archived Entities

ClaimCenter 6.0.5 and newer include a file-based archiving feature. If you implement archiving, and you make custom data model changes, then you can upgrade the archived XML using the `IDatabaseUpgrade` plugin.

Simple data model changes do not require a custom trigger. These include:

- Adding a new entity
- Updating denormalization columns
- Adding editable columns such as `updatetime`
- Adding new columns
- Changing the nullability of a column

More complex transformations or those that could result in loss of data require a version trigger. These include:

- changing a datatype (other than just length)
- migrating data from one table or column to another
- dropping a column
- dropping a table
- renaming a column
- renaming a table

ClaimCenter upgrades an archived entity as the entity is restored.

The `IDatabaseUpgrade` plugin interface includes the following properties:

```
get BeforeUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
get AfterUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
```

You can define custom `ArchivedDocumentUpgradeVersionTrigger` entities to modify archived XML. The `ArchivedDocumentUpgradeVersionTrigger` is an abstract class that you can extend to create your custom triggers.

Define the constructor of your custom `ArchivedDocumentUpgradeVersionTrigger` to call the constructor of the superclass and pass it a numeric value. For example:

```
construct() {
    super(171)
}
```

This numeric value is the extension version to which your trigger applies. If you run the upgrade against a database with a lower extension version, then your custom trigger is called. The current extension version is defined in `modules\cc\config\extensions\extensions.properties`.

Provide an override definition of the `get Description` property to return a `String` that describes the actions of your trigger.

Provide an override definition for the `execute` function to define the actions that you want your custom trigger to make on archived XML.

When the upgrade executes your custom trigger, it wraps each XML entity in an `IArchivedEntity` object. Each typekey is wrapped in an `IArchivedTypekey` object. The upgrade operates on a single XML document at a time.

`ArchivedDocumentUpgradeVersionTrigger` provides the following key operations:

- `getArchivedEntitySet(entityName : String)` – returns an `IArchivedEntitySet` object that contains all `IArchivedEntity` objects of the given type in the XML document.

`IArchivedEntitySet` provides the following key methods:

- `rename(newEntityName : String)` – renames all rows in the set to the new name.
- `delete()` – deletes all rows in the set.
- `search(predicate : Predicate<IArchivedEntity>)` – returns a List of `IArchivedEntity` objects that match the given predicate.
- `create(referenceInfo : String, properties : List<Pair<String, Object>>)` – returns a new `IArchivedEntity` with the given properties.

`IArchivedEntity` provides the following key methods:

- `delete()` – deletes just this row.
- `getPropertyValue(propertyName : String)` – returns the value of the property of the given name. If the property value is a reference to another entity, this method returns an `IArchivedEntity`.
- `move(newEntityName : String)` – moves this to a new entity type of the given name. The type is created if it did not exist. You must add any required properties to the type.
- `updatePropertyValue(propertyName : String, newValue : String)` – updates the property of the given name to the given value.
- `getArchivedTypekeySet(typekeyName : String)` – returns an `IArchivedTypekeySet` object that contains all `IArchivedTypekey` objects in the given typelist.
- `getArchivedTypekey(typelistName : String, code : String)` – Returns an `IArchivedTypekey` representing the typekey.

`IArchivedTypekey` provides the following key method:

- `delete()` – deletes the typekey from the XML.

More methods are available for `IArchivedEntitySet`, `IArchivedEntity` and `IArchivedTypekey`. See the Gosu documentation for a full listing. Generate the Gosu documentation by navigating to the ClaimCenter bin directory and entering the following command:

```
gwcc regen-gosudoc
```

Incremental Upgrade

When ClaimCenter archives an entity, it records the current data model version on the entity. ClaimCenter upgrades an archived entity as the entity is restored. The upgrader executes the necessary archive upgrade triggers incrementally on each archived XML entity, according to the data model version of the archived entity.

An entity archived at one version might not be restored and upgraded until several intermediate data model upgrades have been performed. Therefore, do not delete your custom upgrade triggers.

Consider the following situation. Note that this example is for demonstration purposes only. The version numbers included do not represent actual ClaimCenter versions but are included to explain the incremental upgrade process. Each '+' after a version number indicates a custom data model change.

Guidewire data model changes

v6.0.0 – Entity does not have column X.

v6.0.1 – Guidewire adds column X to the entity with a default value of 100.

v6.0.2 – Guidewire updates the default value of column X to 200.

Implementation #1 upgrade path

v6.0.0+ – Start with version 6.0.0 data model with custom changes.

v6.0.0++ – More custom data model changes.

v6.0.0+++ – More custom data model changes.

v6.0.2+ – column X introduced with a default value of 200.

Result of upgrade of a claim archived at v6.0.0+: column X has value 200.

In this situation, version 6.0.1 was skipped in the upgrade path. Therefore, column X is added with the default value of 200 that it has in version 6.0.2.

Implementation #2 upgrade path

v6.0.0+ – Start with version 6.0.0 data model with custom changes.

v6.0.0++ – More custom data model changes.

v6.0.1+ – column X introduced with a default value of 100.

v6.0.2+ – column X already exists.

Result of upgrade of a claim archived at v6.0.0+: column X has value 100.

In this situation, column X is added in version 6.0.1 with the default value of 100. During the upgrade from version 6.0.1 to version 6.0.2, column X already exists. Therefore, the upgrader does not add the column. A custom trigger would be required to update the value of X from 100 to 200 during the upgrade from version 6.0.1 to 6.0.2.

Disabling the Scheduler

Before you start the server to upgrade the database, disable the scheduler for batch processes and work queues. Disabling the scheduler prevents batch processes and work queues from launching immediately after the database upgrade.

To disable the scheduler

1. Open the ClaimCenter 6.0.8 `config.xml` file in a text editor.
2. Set the `SchedulerEnabled` parameter to `false`.

```
<param name="SchedulerEnabled" value="false"/>
```
3. Save `config.xml`.

After you have successfully upgraded the database, you can enable the scheduler by setting `SchedulerEnabled` to `true`. This can be accomplished by performing the database upgrade using a WAR or EAR file that has the `SchedulerEnabled` parameter to `false`. After the upgrade is complete and verified, stop the server and deploy a new WAR or EAR file that differs from the first only by having `SchedulerEnabled` set to `true`. Finally, restart the server to activate the scheduler.

Suspending Message Destinations

Suspend all event message destinations before you upgrade the database to prevent ClaimCenter from sending messages until you have verified a successful database upgrade.

To suspend message destinations

1. Start the ClaimCenter server for the pre-upgrade version.
2. Log in to ClaimCenter with an account that has administrative privileges, such as the superuser account.
3. Click the **Administration** tab.
4. Click **Event Messages**.
5. Select the check box to the left of the **Destination** column to select all message destinations.

6. Click Suspend.

Resume messaging after you have verified a successful database upgrade.

Configuring the Database Upgrade

You can set parameters for the database upgrade in the ClaimCenter 6.0.8 `config.xml` file. The `<database>` block in `config.xml` contains parameters for database configuration, such as connection information. The `<database>` block contains an `<upgrade>` block that contains configuration information for the overall database upgrade. The `<upgrade>` block also contains a `<versiontriggers>` element for configuring general version trigger behavior and can contain `<versiontrigger>` elements to configure each version trigger.

This topic describes the parameters you can set for the database upgrade. For general database connection parameters, see “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Configuring Column Removal on Oracle

The database upgrade removes some columns. For Oracle, you can configure whether the removed columns are dropped immediately or are marked as unused. Marking a column as unused is a faster operation than dropping the column immediately. However, because these columns are not physically dropped from the database, the space used by these columns is not released immediately to the table and index segments. You can drop the unused columns after the upgrade during off-peak hours to free the space. Or, you can configure the database upgrade to drop the columns immediately during the upgrade. By default, the ClaimCenter database upgrade marks columns as unused.

To configure the ClaimCenter upgrade to drop columns immediately during the upgrade, set the `oracleMarkColumnsUnused` attribute of the `<upgrade>` block to `false`. For example:

```
<database ...>
...
  <upgrade oracleMarkColumnsUnused="false">
    ...
  </upgrade>
</database>
```

Configuring Index Creation Parallelism on Oracle

You can configure the database upgrade to create indexes in parallel on Oracle databases. This can potentially reduce the time to upgrade the database if there are sufficient resources available on the database server. By default, indexes are not created in parallel on Oracle. Enable parallel index creation by setting the `createIndexInParallel` attribute of the `<upgrade>` block in the `<database>` block of `config.xml`. For example:

```
<database ...>
...
  <upgrade createIndexInParallel="2">
    ...
  </upgrade>
</database>
```

Set a numeric value for `createIndexInParallel` to specify the degree of parallelism for index creation during the upgrade. Or, set `value="default"` to have Oracle automatic parallel tuning determine the degree to use. Oracle determines the degree based on the number of CPUs and the value set for the Oracle parameter `PARALLEL_THREADS_PER_CPU`.

Enabling Collection of Tablespace Usage and Object Size

To enable collection of tablespace usage and object size data on Oracle, set the `collectstorageinstrumentation` attribute of the `<upgrade>` block to `true`. For example:

```
<database ...>
...
```

```
<upgrade collectstorageinstrumentation="true">
...
</upgrade>
</database>
```

A value of `true` enables ClaimCenter to collect tablespace usage and size of segments such as tables, indexes and LOBs (large object binaries) before and after the upgrade. The values can then be compared to find the utilization change caused by the upgrade.

Enabling Oracle Logging

You can enable logging of direct insert and create index operations during the database upgrade by setting `allowUnloggedOperations` to `false` in the `<upgrade>` block. For example:

```
<database ...>
...
<upgrade allowUnloggedOperations="false">
...
</upgrade>
</database>
```

The default is to run these statements with the `Nologging` option. Although Guidewire recommends that you backup the database before and after the upgrade, there could be reasons to log all operations. Some examples include Reporting, Disaster Recovery through Standby databases and Oracle Dataguard. To enable logging of direct insert and create index operations, set `allowUnloggedOperations` to `false`.

Storing Temporary Sort Results in tempdb

For SQL Server databases, you can specify to store temporary sort results in `tempdb` by setting the `sqlserverCreateIndexSortInTempDB` attribute of the `upgrade` block to `true`. By using `tempdb` for sort runs, disk input and output is typically faster, and the created indexes tend to be more contiguous. By default, `sqlserverCreateIndexSortInTempDB` is `false` and sort runs are stored in the destination filegroup.

If you set `sqlserverCreateIndexSortInTempDB` to `true`, you must have enough disk space available to `tempdb` for the sort runs, which for the clustered index include the data pages. You must also have sufficient free space in the destination filegroup to store the final index structure, because the new index is created before the old index is deleted. Refer to <http://msdn.microsoft.com/en-us/library/ms188281.aspx> for details on the requirements to use `tempdb` for sort results.

Adjusting Commit Size for Encryption

You can adjust the commit size for rows requiring encryption by setting the `encryptioncommitsize` attribute to an integer in the `<upgrade>` block. For example:

```
<database ...>
...
<upgrade encryptioncommitsize="5000">
...
</upgrade>
</database>
```

If ClaimCenter encryption is applied on one or more attributes, the ClaimCenter database upgrade commits batches of encrypted values. The upgrade commits `encryptioncommitsize` rows at a time in each batch. The default value of `encryptioncommitsize` varies based on the database type. For Oracle, the default is 10000. For SQL Server, the default is 100.

Specifying Filegroup to Store Sort Results for Clustered Indexes

For SQL Server databases, a version trigger recreates non-clustered backing indexes for primary keys as clustered indexes. This change improves the performance of claim archiving and claim purging operations.

Before recreating the indexes, the version trigger automatically drops (and later rebuilds) any referencing foreign keys and drops any clustered indexes on tables with a primary key.

If you are using filegroups, the upgrade recreates the clustered index in the OP filegroup. By default, the upgrade also stores the intermediate sort results that are used to build the index in the OP filegroup. You can configure the upgrade to instead use the tempdb filegroup for the intermediate sort results.

If you want the upgrade to store the intermediate sort results in the tempdb filegroup, set the `sqlserverCreateIndexSortInTempDB` attribute of the upgrade element to `true`.

```
<database ...>
...
<upgrade sqlserverCreateIndexSortInTempDB="true" />
...
</upgrade>
</database>
```

This option increases the amount of temporary disk space that is used to create an index. However, it might reduce the time that is required to create or rebuild an index when tempdb is on a different set of disks from that of the user database.

By default, `sqlserverCreateIndexSortInTempDB` is `false`.

Configuring Version Trigger Elements

The database upgrade executes a series of version triggers that make changes to the database to upgrade between versions. You can set some configuration options for version triggers in `config.xml`. Normally, the default settings are sufficient. Change these settings only while investigating a slow database upgrade.

The `<database>` element in `config.xml` contains an `<upgrade>` element to organize parameters related to database upgrades. Included in the `<upgrade>` element is a `<versiontriggers>` element, as shown below:

```
<database ...>
  <param ... />
  <upgrade>
    <versiontriggers dbmsperfinfothreshold="600" degreeofparallelismforinsertselects="2"/>
  </upgrade>
</database>
```

The `<versiontriggers>` element configures the instrumentation of version triggers. This element has two attributes: `dbmsperfinfothreshold` and `degreeofparallelismforinsertselects`.

The `dbmsperfinfothreshold` attribute specifies for each version trigger the threshold after which the database upgrader gathers performance information from the database. You specify `dbmsperfinfothreshold` in seconds, with a default of 600. If a version trigger takes longer than `dbmsperfinfothreshold` to execute, ClaimCenter:

- queries the underlying database management system (DBMS).
- builds a set of html pages with performance information for the interval in which the version trigger was executing.
- includes those html pages in the upgrader instrumentation for the version trigger.

You can completely turn off the collection of database snapshot instrumentation for version triggers by setting the `dbmsperfinfothreshold` to 0 in `config.xml`.

The `degreeofparallelismforinsertselects` attribute overrides a global default value (2) for the degree of parallelism for all tables involved in insert and select statements in version triggers that benefit from parallelism. This allows users to take advantage of their hardware when doing an upgrade. This attribute is only applicable for Oracle. Set `degreeofparallelismforinsertselects` to 1 if running on a single CPU machine.

The `<versiontriggers>` element can contain optional `<versiontrigger>` elements for each version trigger. Each `<versiontrigger>` element can contain the following attributes.

Attribute	Type	Description
<code>name</code>	String	The case-insensitive name of a version trigger.
<code>recordcounters</code>	Boolean	Controls whether the DBMS-specific counters are retrieved at the beginning and end of the use of the version trigger. Default is <code>false</code> . If <code>true</code> , then ClaimCenter retrieves the current state of the counters from the underlying DBMS at the beginning of execution of the version trigger. If the execution of the version trigger exceeds the <code>dbmsperfinfothreshold</code> , then ClaimCenter retrieves the current state of the counters at the end of the execution of the version trigger. ClaimCenter writes differences to the DBMS-specific instrumentation pages of the upgrade instrumentation.
<code>extendedquerytracingenabled</code>	Boolean	Oracle only. Controls whether or not to enable extended sql tracing (Oracle event 10046) for the SQL statements that are executed by the version trigger. Default is <code>false</code> . The output can be very useful when debugging certain types of performance problems. Trace files that are generated only exist on the database machine. They are not integrated into the upgrade instrumentation.
<code>queryoptimizertracingenabled</code>	Boolean	Oracle only. Controls whether or not to enable query optimizer tracing (Oracle event 10053) for the SQL statements that are executed by the version trigger. Default is <code>false</code> . The output can be very useful when debugging certain types of performance problems. Trace files that are generated only exist on the database machine. They are not integrated into the upgrade instrumentation.
<code>updatejoinhintbody</code>	String	Oracle only. Used to specify the body of an optimizer hint (without the surrounding <code>/*+ */</code>) for the upgrade of a join in the version trigger. If the update has a default override and the user specifies the empty string <code>""</code> as the value of the attribute, then no hint will be specified.

Starting the Server to Begin Automatic Database Upgrade

The database upgrade is an automatic process that occurs as you start the server with the upgraded configuration. The database upgrade normally completes in a few hours or less.

If the database upgrade stops before completing, then restore your database from the backup, correct any issues reported, and repeat the database upgrade.

IMPORTANT Before starting the upgrade, update database server software and operating systems as needed to meet the installation requirements of the target version. 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>.

WARNING Except for your first database upgrade trials, do not start the server until you have upgraded all rules. Otherwise, default validation rules execute. This could strand objects at a high validation level and make it impossible to edit parts of the object.

WARNING The database upgrade runs a series of version checks prior to making any changes. If any of these checks fail, the upgrade aborts and reports an error message. You can fix the issue, create an updated backup of the database and attempt the upgrade again without restoring from a backup. However, if you experience a failure during the version triggers or upgrade steps portion of the upgrade, refresh the database from a backup before attempting the upgrade again.

Test the Database Upgrade

Prior to attempting the database upgrade on a full-production database clone, test the database upgrade by doing the following:

1. Connected to the built-in H2 database, successfully start the built-in Quickstart (Jetty) application server with a merged configuration data model, including merged extensions, datatypes, fieldvalidators, and so forth.
2. Connected to an empty database on an Oracle or SQL Server database server, successfully start the Quickstart application server from the preceding step.
3. Connected to a restored backup of a production clone, start either the same Quickstart server from the preceding step or a third-party application server with your custom configuration.

Integrations and Starting the Server

Disable all integrations during the automatic database upgrade. Integration points might require updates due to changes in Guidewire APIs. See the *ClaimCenter New and Changed Guide* for specifics.

It is not necessary to have completely migrated integrations before attempting to start the server for the first time. If you have integrations that rely on non-Guidewire applications, do not expect these integrations to work the first time you start the server.

Understanding the Automatic Database Upgrade

As the database upgrade proceeds, it logs messages to the console as well as the log file describing its progress. The database upgrade process requires thousands of steps, divided into three phases. Due to the relational nature of a database, these phases must execute in a specific order for the upgrade to succeed.

During the first phase, the upgrader uses a set of *version triggers* to determine the first actions that need to take place. The database upgrader requires version triggers in order to perform the following types of tasks:

- changing a datatype (other than just length)
- migrating data from one table or column to another
- dropping a column

- dropping a table
- renaming a column
- renaming a table

Specific version triggers are described in this topic.

Many version triggers have version checks associated with them. These checks ensure that the database is ready for the associated version trigger. The database upgrade runs all checks before running any version triggers. If a check detects a problem, it reports the issue, including a sample SQL query to find specific problematic records. If a version check discovers an issue, the database upgrade stops before any version triggers are run. Therefore, it is not necessary to restore the database from a backup if a version check reports an error. Correct the issue and then create a new backup of the database. Then, if you encounter errors after the version check stage, you can restore a version of your database with the issue reported by the version check resolved.

In the second phase, the upgrader compares the target data model and the current database to determine how they differ. The upgrader makes changes to the database that do not require a version trigger during this phase.

Following this process, the third phase runs a subsequent set of version triggers. These triggers create actions that must be run last due to a dependency on an earlier phase.

After the database upgrade concludes, it reports issues that the upgrader encountered and did not complete.

You are responsible for correcting these issues. This might involve modifying the data model or altering the table manually. If you do not correct them, the next time you start the server you do *not* see a message that the database and the data model are out of sync. You must then use the `system_tools` command to verify the database schema.

Note: Given the complexity of database upgrade, Guidewire does not expose specific upgrade actions/steps to clients either in SQL or Java form. Any manual attempts to recreate or control the upgrade process can result in problems in the ClaimCenter database. Recovery from such attempts is not supported.

Version Trigger Descriptions

The database upgrade uses version triggers to perform the actions described by sections within this topic. If a version trigger has an associated version check, the check is described with the trigger. Review these descriptions to familiarize yourself with some of the changes and to understand version checks. If a version check reports an issue, review the error message and consult the description of the relevant version trigger for more information.

Converting Primary Key Indexes to Clustered Indexes

For SQL Server databases, this version trigger recreates non-clustered backing indexes for primary keys as clustered indexes. This change improves the performance of claim archiving and claim purging operations.

Before recreating the indexes, the version trigger automatically drops (and later rebuilds) any referencing foreign keys and drops any clustered indexes on tables with a primary key.

If you are using filegroups, the upgrade recreates the clustered index in the OP filegroup. By default, the upgrade also stores the intermediate sort results that are used to build the index in the OP filegroup. You can configure the upgrade to instead use the tempdb filegroup for the intermediate sort results.

If you want the upgrade to store the intermediate sort results in the tempdb filegroup, set the `sqlserverCreateIndexSortInTempDB` attribute of the upgrade element to `true`.

```
<database ...>
...
  <upgrade sqlserverCreateIndexSortInTempDB="true" />
  ...
</upgrade>
</database>
```

This option increases the amount of temporary disk space that is used to create an index. However, it might reduce the time that is required to create or rebuild an index when tempdb is on a different set of disks from that of the user database.

By default, `sqlserverCreateIndexSortInTempDB` is `false`.

Renaming ClaimInfo Columns

This step renames `cc_ClaimInfo.ArchiveFailure` to `cc_ClaimInfo.ArchiveFailureID` and `cc_ClaimInfo.ArchiveFailureDetails` to `cc_ClaimInfo.ArchiveFailureDetailsID`.

Enforcing Contact Address Uniqueness

This trigger eliminates duplicate Address records in `ContactContact.Address` and `Contact.PrimaryAddress`. For duplicate Address records, the trigger creates a unique Address record and relinks the duplicate references to the unique Address.

Changing Datatype of ArchiveTransitionRec.InternalDesc

This step alters the datatype of the `cc_archivetransitionrec.internaldesc` column from `Text` to `MediumText`.

Changing Precision of Metric Percent Columns

This step alters the precision of the following metric percent columns to eight, with a scale of zero:

- `cc_claimmetric.PercentValue`
- `cc_claimmetriclimit.PercentRedValue`
- `cc_claimmetriclimit.PercentTargetValue`
- `cc_claimmetriclimit.PercentYellowValue`
- `cc_exposuremetric.PercentValue`
- `cc_exposuremetriclimit.PercentRedValue`
- `cc_exposuremetriclimit.PercentTargetValue`
- `cc_exposuremetriclimit.PercentYellowValue`

This trigger only runs if the `cc_claimmetric` table exists already, so it only runs if the database is being upgraded from ClaimCenter 6.0. For database upgrades from earlier versions, the metric percent columns are defined in the data model with the correct precision.

Adding Encryption to Claim Snapshots

ClaimCenter 6.0.8 provides encryption of sensitive data, such as tax IDs, on claim snapshots.

This step adds an `EncryptionVersion` integer column to `cc_claimsnapshot`. As of ClaimCenter 6.0.1, you can create multiple versions of `IEncryption` plugins. The `cc_claimsnapshot.EncryptionVersion` 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 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.

During the upgrade to ClaimCenter 6.0.8, an upgrade trigger increments the global encryption version number. So, when the Encryption Upgrade work queue runs, it will encrypt fields in the snapshot. This trigger does not fire if encryption is not enabled, and there are no encrypted fields defined on the claim snapshot.

For more information about the encryption plugin, see “Encryption Integration” on page 401 in the *Integration Guide*.

For more information about the Encryption Upgrade work queue, see “Batch Processes and Distributed Work Queues” on page 134 in the *System Administration Guide*.

Refactoring Claim and Exposure Staging Tables

This step drops the `ccst_claim` and `ccst_exposure` tables. ClaimCenter 6.0.2 and higher define `Claim.ClaimantDenorm`, `Claim.InsuredDenorm` and `Exposure.ClaimantDenorm` with the attribute `overwrittenInStagingTable` set to `true`. ClaimCenter automatically populates these columns when you load records into the staging tables. Previous versions of ClaimCenter populated these columns after the staging table records were moved into the main repository. In some cases, this caused performance issues. To remedy this, ClaimCenter 6.0.2 and higher populate these columns when you load the staging table rather than when you move data from the staging tables to the main repository.

Claim and exposure entities are defined as loadable, so ClaimCenter creates new `ccst_claim` and `ccst_exposure` tables when the server starts.

Cleaning up Cross-Claim ReserveLineWrapper.ReserveLine References

The trigger finds instances in which the `BulkInvoiceItemInfo` of a `ReserveLineWrapper` points to a different `Claim` than the `Claim` pointed to by the `ReserveLine`. The trigger updates those `ReserveLineWrapper` entities by repointing a given `ReserveLineWrapper` to a particular `ReserveLine` on its own `Claim`.

The trigger selects the `ReserveLine` belonging to the `Transaction` of the transferred Check. Typically, a `BulkInvoice` Check has only one `Transaction`. However, in this case, cross-claim linkage occurs when the `BulkInvoice` Check is transferred between `Claims`. In that case, the Check has two `Transactions`, an offset on the first `Claim` and an onset on the second. The trigger selects the onset `Transaction` on the original `Claim`.

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.

Dropping Unused Columns on Oracle

By default, the ClaimCenter database upgrade on Oracle marks some columns as unused rather than dropping the columns. You can configure this behavior by setting the `oracleMarkColumnsUnused` parameter to `false` before running the database upgrade. This parameter is within the `<upgrade>` block of the `<database>` block of `config.xml`. If you set `oracleMarkColumnsUnused` to `false` before the upgrade, the upgrade already dropped removed columns. In that case, you do not need to perform the procedure in this section to drop unused columns.

Marking a column unused is a faster operation than dropping a column. Because these columns are not physically dropped from the database, the space used by these columns is not released immediately to the table and index segments. You can drop the unused columns after the upgrade during off-peak hours to free the space. ClaimCenter does not have to be shutdown to perform this maintenance task. You can drop all unused columns in one procedure, or you can drop unused columns for individual tables.

To drop all unused columns

1. Create the following Oracle procedure to purge all unused columns:

```
DECLARE
  dropstr VARCHAR2(100);
  CURSOR unusedcol IS
    SELECT table_name
      FROM user_unused_col_tabs;
BEGIN
  FOR tabs IN unusedcol LOOP
    dropstr := 'alter table '
              || tabs.table_name
              || ' drop unused columns';
    EXECUTE IMMEDIATE dropstr;
  END LOOP;
END;
```

2. Run the procedure during a period of relatively low activity.

To drop unused columns for a single table

1. Start the server to run the schema verifier. The schema verifier runs each time the server starts. If there are unused columns, the schema verifier reports a difference between the physical database and the data model. The schema verifier reports the name of each table and provides an SQL command to remove unused columns from each table.
2. Run the SQL command provided by the schema verifier. This command has the following format:

```
ALTER TABLE tableName DROP UNUSED COLUMNS
```

Setting Claim Descriptions

ClaimCenter 6.0 provides the option of using claim-level Insurance Services Office (ISO) messaging. Previous versions of ClaimCenter sent exposures to ISO. With ClaimCenter 6.0, you have the option of sending ISO messages at either the claim or exposure level. See “ISO Changes in ClaimCenter 6.0” on page 87 in the *New and Changed Guide*.

New ISO validation rules require that a description is set on a claim if the claim is to be sent to ISO at the claim level. You will not be able to edit claims that lack a description until you set the description. Any batch processes that modify these claims or their objects, such as activities, will fail until you set the description. The failed batch process will report the following validation error:

```
[java] com.guidewire.pl.system.bundle.validation.EntityValidationException:
error:entity.Claim.Description/The claim's description must not be null
```

If a claim has already been sent to ISO at the exposure level, then it will never be sent at the claim level. Therefore, the description requirement only applies to claims that have not had any exposures sent to ISO.

If the following conditions are true, check for claims with null descriptions and add a description:

- You have had ISO enabled.
- You want to switch to claim-level ISO messaging.
- You have claims which have null descriptions and have not yet had any of their exposures sent to ISO.

You can check for claims that will cause this validation issue with the following SQL query:

```
SELECT claimnumber FROM cc_claim c
WHERE c.description IS NULL
AND NOT EXISTS (SELECT * FROM cc_exposure e
                WHERE e.ClaimID = c.ID AND e.ISOSendDate IS NOT NULL)
```

If this query returns any claim numbers, set a description for each corresponding claim.

Final Steps After The Database Upgrade is Complete

This section describes the checks and procedures to run after you have completed the upgrade procedure and migration of configurations and integrations. The processes and checks in this section provide you with a benchmark of the new system. Completing these steps is particularly important to going live in a production environment.

IMPORTANT For procedures referenced below that use utilities in `toolkit/bin` on the pre-upgrade configuration, use the utility from `admin/bin` in the target configuration following the upgrade. You do not need to generate the toolkit in the target configuration to use these utilities.

Use these procedures to revalidate the database:

- “Validating the Database Schema” on page 50
- “Checking Database Consistency” on page 51
- “Creating a Data Distribution Report” on page 51
- “Generating Database Statistics” on page 52
- “Backing up the Database After Upgrade” on page 74

Backing up the Database After Upgrade

Finally, before going live, back up the upgraded database. This provides you with a snapshot of the initial upgraded data set, if an unanticipated event occurs just after going live.

Upgrading from 5.0.x

This part describes how to perform an upgrade from ClaimCenter 5.0.x to 6.0.8.

If you are upgrading from ClaimCenter 6.0.x, see “Upgrading from 6.0.x” on page 23 instead.

If you are upgrading from ClaimCenter 4.0.x, see “Upgrading from 4.0.x” on page 173 instead.

This part includes the following topics:

- “Upgrading the ClaimCenter 5.0.x Configuration” on page 77
- “Upgrading the ClaimCenter 5.0.x Database” on page 117
- “Upgrading Integrations and Gosu from 5.0.x” on page 167

Upgrading the ClaimCenter 5.0.x Configuration

This topic describes how to upgrade the ClaimCenter and ContactCenter configuration from version 5.0.x.

If you are upgrading from a 4.0.x version, see “Upgrading the ClaimCenter 4.0.x Configuration” on page 175 instead.

If you are upgrading from a 6.0.x version, see “Upgrading the ClaimCenter 6.0.x Configuration” on page 25 instead.

This topic includes:

- “Overview of ContactCenter Upgrade” on page 78
- “Obtaining Configurations and Tools” on page 78
- “Creating a Configuration Backup” on page 82
- “Updating Infrastructure” on page 82
- “Deleting Target Configuration Module” on page 82
- “Removing Size Attribute from Integer and Money Datatypes” on page 82
- “Merging the ClaimCenter Configuration” on page 82
- “Upgrading Rules to ClaimCenter 6.0.8” on page 108
- “Translating New Display Properties and Typecodes” on page 110
- “Modifying PCF files, Rules and Libraries for Unused Contact Subtypes” on page 111
- “Converting Velocity Templates to Gosu Templates” on page 111
- “Validating the ClaimCenter 6.0.8 Configuration” on page 114
- “Building and Deploying ClaimCenter 6.0.8” on page 115

Overview of ContactCenter Upgrade

The upgrade process for ContactCenter is almost precisely the same as for ClaimCenter. Follow the procedures in this guide to upgrade ContactCenter. Procedures specific to the ContactCenter upgrade are indicated as such. Some information in this guide is specific to ClaimCenter and does not apply to ContactCenter. However, the basic upgrade procedure is the same.

Obtaining Configurations and Tools

Configuration refers to everything related to the application except the database. This includes configuration files such as typelists and PCF files, the file structure, web resources, Gosu classes, rules, plugins, libraries, localization files, and application server files.

The upgrade process involves three configurations. This guide defines and refers to these configurations as **base**, **customer**, and **target**.

Base: The unedited, original configuration on which you based your customer configuration. The base configuration is included in directories within `/modules` other than `/configuration`.

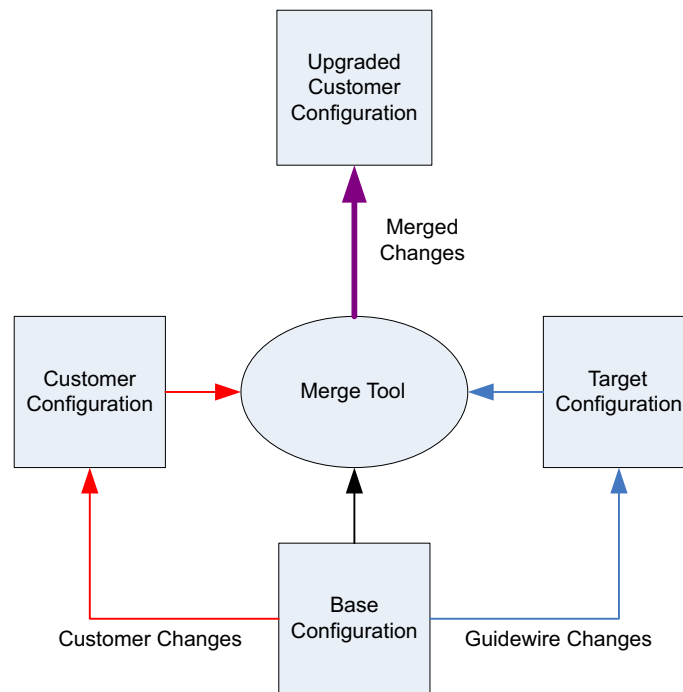
Customer: The configuration you are now using and will upgrade. This is the base configuration of the ClaimCenter version that you currently run with your custom configuration applied. The `/modules/configuration` directory contains custom changes.

Target: The unedited, original configuration of ClaimCenter 6.0.8 on which your upgraded configuration will be based. Guidewire grants you access to the Guidewire Resource Portal, from which you download the target configuration ZIP file. Unzip the target ClaimCenter 6.0.8 configuration into another directory. Download the latest patch release for the target version that you are downloading. Follow the instructions with the patch release to install it after you unzip the target version. Do not make any modifications to the target configuration prior to completing the configuration upgrade. Do not start Guidewire Studio for the target configuration until you have completed the configuration upgrade.

IMPORTANT Set all files in the base, customer, and target configurations to writable before beginning the upgrade.

The following figure shows how you use these configurations to create a merged configuration. The merged configuration combines your changes to the original base configuration (the customer configuration) and Guidewire

changes to the base configuration (the target configuration). The original base configuration provides a basis for comparison.



Viewing Differences Between Base and Target Releases

To view an inventory of the differences between the base release and the target release, download and carefully review the *Upgrade Diffs Report* from the Guidewire Resource Portal.

1. Open a browser to https://guidewire.custhelp.com/app/projectcenter/upgrades/diff_reports/.
2. Click ClaimCenter or ContactManager.
3. Click **Upgrade From** *base version*.
4. Click **Upgrade To 6.0.8**.
5. Download the three files into the same folder.
6. Open `Report.html` with a browser.

Specifying Configuration and Tool Locations

The ClaimCenter 6.0.8 Configuration Upgrade Tool depends on the following tools:

- **Text Editor:** An ASCII text editor you use to edit programs and similar files, for example, Notepad, WordPad or Textpad. This editor must not put additional characters in files, as Word does.
- **Merge Tool:** An editor which displays two or three versions of a file, highlights the differences between them, and allows you to perform basic editing functions on them. Also known as a “diff tool.” Examples include Araxis Merge Professional and Beyond Compare 3 Professional. Configure the merge tool to ignore end of line characters to reduce the number of potential false positives during the configuration upgrade step.

IMPORTANT The merge tool that you use must support three-way file comparison and merging. During the configuration upgrade, for some files you will need to compare three versions: the original base version, the new version and your customized version.

The Configuration Upgrade Tool needs the location of the ClaimCenter environment that you will upgrade. The tool stores all versions of files to be merged and merge results in a /tmp directory that it creates within the target environment. Define paths to the configuration and tools in the /ClaimCenter/modules/ant/upgrade.properties file of the target ClaimCenter 6.0.8 environment. Remove the pound sign, '#', preceding each property to uncomment the line and then specify values. Use double backslashes in paths. For example, C:\\ClaimCenter. You do not need to use quotes for paths that include spaces.

The following properties are configurable in `upgrade.properties`.

Property	Description
<code>upgrader.priorversion.dir</code>	Path to the top-level ClaimCenter directory of the current customer environment. This directory contains <code>/bin</code> and other ClaimCenter directories.
<code>upgrader.editor.tool</code>	Path to an executable editing tool.
<code>upgrader.diff.tool</code>	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool, used for two-way merges. If your difference editor supports both two and three-way merges, you can use the same value for <code>upgrader.diff.tool</code> and <code>upgrader.merge.tool</code> .
<code>upgrader.merge.tool</code>	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool, used for three-way merges. The merge tool specified for <code>upgrader.merge.tool</code> must support three-way file comparison and merging. If your difference editor supports both two and three-way merges, you can use the same value for <code>upgrader.diff.tool</code> and <code>upgrader.merge.tool</code> . You might need to configure the display of your merge tool to show three panels.
<code>upgrader.merge.tool.arg.order</code>	The display order, from left to right, for versions of a file viewed in the difference editor tool specified by <code>upgrader.merge.tool</code> . By default, the tool displays, left to right, the versions of a file in this order: <code>NewFile OldFile ConfigFile</code> in which: <code>NewFile</code> is the unmodified target version provided with ClaimCenter 6.0.8. <code>OldFile</code> is the original base version. <code>ConfigFile</code> is your configured version. The order of these values controls the display order in the difference editor tool. If the tool displays just two versions, it uses the same relative order. By default, the display order places the old base version of a file in the center. The old base version is the common ground between the new uncustomized version and the old customized version. Guidewire changed the old base version to create the new target version, and you changed the old base version to create the customized version in your configuration. With the old base version in the center, you can incorporate both sets of changes to create a customized target version. The default order enables you to merge changes from the left and right to the center and save the merged version. If you use another difference editor, you might need a different order to achieve the same result. Samples are shown below for various difference engines: Araxis Merge Professional <code>upgrader.merge.tool.arg.order = NewFile OldFile ConfigFile</code> Beyond Compare 3 Professional <code>upgrader.merge.tool.arg.order = NewFile ConfigFile OldFile</code> P4Merge <code>upgrader.merge.tool.arg.order = OldFile NewFile ConfigFile</code> You might need to configure the display of your merge tool to show three panels.
<code>exclude.pattern</code>	A regular expression pattern for paths of files for the Configuration Upgrade Tool to mark as unmergeable. Typically, you use <code>exclude.pattern</code> to specify source control metadata files. Samples are provided in <code>upgrade.properties</code> for CVS and SVN.

Creating a Configuration Backup

Prepare the environment so that you can make a total recovery of the original installation if you run into problems during the upgrade.

Guidewire recommends that you track ClaimCenter configuration changes in a source code control system. Before upgrading, have a labeled version of your entire ClaimCenter installation. A labeled version is a named collection of file revisions.

As an even stronger precaution, make a backup of the same installation directories.

Updating Infrastructure

Before starting the upgrade, have the supported server operating systems, application server and database software, JDK, and client operating systems for the target version. See “Installation Environments Overview” on page 12 in the *Installation Guide* for supported versions for ClaimCenter 6.0.8.

Deleting Target Configuration Module

Delete all files and folders in the `modules\configuration` folder of the target ClaimCenter 6.0.8 configuration except `gwmodule.xml`. These files are included with new installations to help rapidly set up a QuickStart environment. You do not need these files to upgrade ClaimCenter, except `gwmodule.xml`, which is required by Studio.

During the manual configuration upgrade process, the Configuration Upgrade Tool copies files that you merge into the configuration module. If you do not delete the contents of the target `modules\configuration` folder, then the Configuration Upgrade Tool locates any file within the folder and considers it merged. However, the file is from the default target configuration rather than an actual merged file.

Removing Size Attribute from Integer and Money Datatypes

Before you run the ClaimCenter 6.0 Configuration Upgrade Tool, check your `extensions.xml` file for any column extensions of type `integer` or `money`. If you have `integer` or `money` type columns defined, remove any `size` attribute from the extension column.

The `size` attribute only applies to `varchar` and `text` type columns. Prior versions of ClaimCenter ignored the error, but ClaimCenter 6.0 reports an error that unknown parameters are specified on the entity when you regenerate the toolkit.

You can fix the issue after the upgrade by removing any `logicalSize` `columnParam` from `integer` and `money` type extensions. However, it is typically simpler to do so before the upgrade while the extensions are contained in a single file.

Merging the ClaimCenter Configuration

To upgrade your configuration, merge Guidewire changes to the base configuration with your changes. The *Configuration Upgrade Tool*, provided by Guidewire with the target configuration, facilitates this process.

The Configuration Upgrade Tool requires two tools: a merge tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, and a text editor. Configure the merge tool to ignore end-of-line characters to reduce the number of potential false positives during the configuration upgrade step.

IMPORTANT The merge tool that you use must support three-way file comparison and merging. During the configuration upgrade, for some files you will need to compare three versions: the original base version, the new version and your customized version.

The Configuration Upgrade Tool performs a series of automated steps and then opens an interface that you use for the manual merge process.

Guidewire can provide guidance on using the Configuration Upgrade Tool in a multi-user environment. Knowledge Base article 571 describes distributing the merge work among users and using a source control management system for this process in a multi-user environment.

See the *ClaimCenter New and Changed Guide* for a description of new features and changes to existing features. Review key data model changes as these changes might impact customizations in your system.

Running the Configuration Upgrade Tool

To launch the Configuration Upgrade Tool

1. Open a command window.
2. Navigate to the `modules/ant` directory of the target configuration.
3. Execute the following command:

```
ant -f upgrade.xml upgrade > upgrade_log.txt
```

You can specify any file to log messages and exceptions.

The Configuration Upgrade Tool first copies the modules of the base environment to a `tmp/cfg-upgrade/modules` directory in the target environment. The base environment is specified by the `upgrader.priorversion.dir` property in `ant/modules/upgrade.properties` in the target environment.

The Configuration Upgrade Tool then performs a number of automated steps, described later in this topic. Once the tool completes the automated steps, it opens a user interface. The interface opens whether the automated steps were successful or not. Review the log file or console before proceeding with the manual merge process.

Restarting the Configuration Upgrade Tool

The Configuration Upgrade Tool stores work in progress by recording which files you have marked resolved in the `accepted_files.lst` file. This file is stored in the merge directory of the target environment. You can close the interface and restart it later without losing your work in progress.

If you do want to start the upgrade over, use the `clean` command to empty the working directories.

```
ant -f upgrade.xml clean
```

WARNING If you empty the `tmp` directory after beginning to merge, you lose all completed merges that you have not resolved and moved into the target configuration directory.

Configuration Upgrade Tool Automated Steps

The Configuration Upgrade Tool prepares for the manual configuration merge process by performing a number of automated steps. Review these steps before proceeding with the configuration merge. Understanding these automated steps helps to understand some file changes you will see when merging the configuration. Finally, some steps might require manual intervention if there is an issue.

Moving Configuration Files

This step moves certain configuration files and directories to new locations. The following table lists old and new locations.

Old location	New location
config/import/security-zones.xml	config/security/security-zones.xml
config/currency/currencies.xml	config/locale/currencies.xml
config/geodata/address-config.xml	config/locale/address-config.xml
config/geodata/zone-config.xml	config/locale/zone-config.xml
config/resources/classes/	gsrc/
config/resources/tests/	gtest/
config/extensions/fieldvalidators.xml	config/fieldvalidators/fieldvalidators.xml

Upgrading Scheduler Configuration

This step upgrades scheduler-config.xml. The process names are updated as described in the following table.

Old process name	New process name
abcontactscoring	ABContactScoring
abgeocode	ABGeocode
accountinactivity	AccountInactivity
activityescalation	ActivityEsc
advanceexpiration	AdvanceExpiration
agencysuspendpayment	AgencySuspensePayment
audittask	AuditTask
automaticdisbursement	AutomaticDisbursement
boundpolicyexception	BoundPolicyException
branchsnapshot	BranchSnapshot
bulkinvoicesescalation	BulkInvoiceEsc
bulkinvoicesworkflow	BulkInvoiceWF
chargeproratatax	ChargeProRataTx
claimexception	ClaimException
closedpolicyexception	ClosedPolicyException
collateraleffective	CollEffective
collateralexpiration	CollExpiration
colleffective	CollEffective
collexpiration	CollExpiration
commissionpayable	CmsnPayable
commissionpayment	CommissionPmt
contactautosync	ContactAutoSync
dashboardstatistics	DashboardStatistics
disbursement	Disbursement
excessfund	ExcessFund
exchangerate	ExchangeRate
financialscal	FinancialsCalc
financialsescalation	FinancialsEsc
fullpaydiscount	FullPayDiscount
geocode	Geocode
groupexception	GroupException
idleclaimexception	IdleClaim

Old process name	New process name
invoice	Invoice
invoicedue	InvoiceDue
jobexpire	JobExpire
letterofcredit	LetterOfCredit
newpayment	NewPayment
openpolicyexception	OpenPolicyException
paymentrequest	PaymentRequest
policyrenewalstart	PolicyRenewalStart
premiumreportingreportdue	PremiumReportReportDue
premiumreportreportdue	PremiumReportReportDue
producerpayment	ProducerPayment
receivableaging	ReceivableAging
releasechargeholds	ReleaseChargeHolds
releasettktholdtypes	ReleaseTktHoldTypes
reviewsync	ReviewSync
statementbilled	StatementBilled
statementdue	StatementDue
statistics	Statistics
statreport	StatReport
suitehistorycleanup	SuiteHistoryCleanup
suspensepayment	SuspensePayment
taccountescalation	TAccountEsc
testtimestats	TestTimeStatistics
troubleticketescalation	TroubleTicketEsc
userexception	UserException
userstatistics	UserStats
userstats	UserStats
workflow	Workflow

Moving Work Queue Configuration

This step moves work-queue configuration elements from `config/config.xml` to `config/workqueue/work-queue.xml`. This step also replaces the name attribute of the work-queue element with `workQueueClass`, according to the following table.

Work queue name	Work queue class
ABGeocode	<code>com.guidewire.ab.domain.geodata.geocode.ABGeocodeWorkQueue</code>
ABContactScoring	<code>com.guidewire.ab.domain.contact.ABContactScoringWorkQueue</code>
ClaimValidation	<code>com.guidewire.cc.domain.claim.impl.ClaimValidationWorkQueue</code>
ContactAutoSync	<code>com.guidewire.pl.system.contactautosync.ContactAutoSyncWorkQueue</code>
Geocode	<code>com.guidewire.pl.domain.geodata.geocode.GeocodeWorkQueue</code>
ReviewSync	<code>com.guidewire.cc.domain.contact.ReviewSyncWorkQueue</code>
Workflow	<code>com.guidewire.pl.system.workflow.engine.monitor.WorkflowDistributedWorkQueue</code>

Upgrading Data Model and Typelist Metadata Files

This step converts entities in data model and typelist metadata files in the `config/metadata` folder of each module to the new format. Each entity element in these files becomes a separate file. The file extension of the new file depends on the entity type.

File extension	Entity types
.eix	internalExtension
.eti	component delegate entity extensionarray nonPersistentEntity subtype viewEntity
.etx	extension viewEntityExtension
.tix	internaltypelistextension
.tti	typelist
.ttx	typelistextension

Data model metadata files were previously named using the format `dm_cc_name.xml`. They now use the format `name.eti`. Data model extension files use the name format `name.etx`. Internal data model extensions use the name format `name.eix`.

Typelist metadata files were previously named using the format `tl_cc_name.xml`. They now use the format `name.tti`. Typelist extension files use the format `name.ttx`. Internal typelist extensions use the name format `name.tix`.

This step also converts `extensions.xml` into separate `entityName.etx` files for each extension. ClaimCenter 6.0 no longer uses the `extensions.xml` file for extension and custom entity definitions. For information on creating entities and extensions to the ClaimCenter data model, see “Modifying the Base Data Model” on page 233 in the *Configuration Guide*.

IMPORTANT This step copies comments from above entity or extension definitions in `extensions.xml`. The upgrade copies everything after the prior `</extension>` or `</entity>` closing tag to the new file for the entity or extension. If you have comments in `extensions.xml`, make sure the comments are placed above or within the entity or extension definition. Then, the upgrade copies the comments to the correct new file.

This step also makes the following changes to the metadata:

- Replaces `<extensionarray>` elements with the basic `<array>` element. The `extensionarrayentity` attribute on these elements has been changed to `arrayentity`. The `extensionarraytable` attribute has been removed from these elements.
- Replaces `<extensionarraytable>` elements with an `<entity>` of type `joinarray`.
- Removes the `table` attribute from `<nonPersistentEntity>`.
- Changes the `long` column type to `longint`.
- Changes the root element name of typelists that are extensions from `<typelist>` to `<typelistextension>`. Removes the `final` attribute from such typelists. Removes the `abstract` and `baseList` attributes.
- Fixes references to base entities. For example, the `Activity` entity used to have a base entity, `ActivityBase`. Prior to ClaimCenter 6.0, you could refer to `Activity` either as `Activity` or `ActivityBase`. Now you can only refer to `Activity` as `Activity`. The upgrader changes any reference to `EntityBase` to `Entity`. For example, the upgrader changes any reference to `ActivityBase` to `Activity`.

- The default for the `setterScriptability` attribute was set to `all` for most fields, but `doesNotExist` for `<onetoone>` elements. Now, it has the same default, `all`, for all fields. The upgrader sets `setterScriptability` to `doesNotExist` for any `<onetoone>` element without an explicit `setterScriptability` attribute defined.
- The `<validationTriggerOverrides>` element has been removed. Instead, these overrides are handled as field override elements. If a `<validationTriggerOverride>` existed for a field, the upgrader replaces it with either a `<foreignkey-override>`, `<array-override>`, or `<onetoone-override>` element, depending on the type of field being overridden. The upgrader adds these overrides to the extension for the owning entity. If no extension exists for that entity, the upgrader creates a new one.
- Replaces the `<delegatesto>` element with `<implementsEntity>` or `<implementsInterface>`, depending on whether the type of delegate being implemented is an entity or a basic java object.
- Removes `<arg>` elements appearing under `<delegatesto>` elements. These were never used and are obsolete.
- Removes the `implements` attribute and replaces it by splitting up the value (comma-separated) and creating `<implementsEntity>` elements for each one.
- Changes foreign key names ending in "ID" by removing the "ID" suffix from the name, and setting the `columnName` attribute to the former name. Fixes references to the old name in the following:
 - the `arrayfield` attribute of `<array>`
 - the `linkField` attribute of `<onetoone>`
 - the `fieldName` attribute of `<validationTriggerOverride>`
 - the `path` attribute of `<viewEntityColumn>`, `<viewEntityName>`, `<viewEntityTypekey>`, and `<viewEntityLink>`
 - the `paths` attribute of `<computedcolumn>` and `<computedtypekey>`
- Removes the obsolete `immutable` attribute from `<foreignkey>`, `<typekey>`, and `<array>`.

Upgrading Column Attributes to Subelements

This step upgrades certain attributes of column entities into separate `columnParam` elements. This step operates on `.eti` and `.etx` files created by the last step in the `config/metadata` and `config/extensions` folders. The following attributes are converted to `columnParam` elements:

- `size`
- `scale`
- `precision`
- `encryption`
- `trimwhitespace`

The `type` attribute and other attributes remain on the column.

For example, the entity:

```
<entity entity="TestOverride" ...>
  <column name="ScaleOverride" precision="5" scale="2" type="decimal"/>
</entity>
```

becomes:

```
<entity entity="TestOverride" ...>
  <column name="ScaleOverride" type="decimal"/>
    <columnParam name="precision" value="5"/>
    <columnParam name="scale" value="2"/>
  </column>
</entity>
```

This step changes the `size` attribute for all column types other than `varchar` to `logicalSize`.

If the column type is `encryption`, this step changes the type to `varchar` and adds the following `columnParam` element:

```
<columnParam name="encryption" value="true"/>
```

Deleting Unused Extensions

This step parses the extension .eti and .etx files created in a previous step and adds the following to the customer extensions directory:

- A <deleteEntity> for any extension entities that were removed.
- Empty <extension> for any entity extensions that were removed. These are extensions that have been removed from entities that remain.
- Empty <viewEntityExtension> for any view entity extensions that were removed.

IMPORTANT If you had deleted an out of the box Contact subtype, such as Adjudicator, check addressbook/contact-sync-config.xml for any references to that entity that will no longer be used. Remove any references to the deleted Contact subtype in contact-sync-config.xml.

Deleting Old Format Metadata Files

This step deletes the old metadata files dm_*, tl_*, extensions.xml, typelists/*.xml, metadataproperties.xml, and platform_metadataproperties.xml from the working directory.

Upgrading Field Validator Configuration

This step moves DataTypes elements from /config/fieldvalidators/fieldvalidators.xml to a separate file: /config/fieldvalidators/datatypes.xml. An earlier step moved fieldvalidators.xml from /config/extensions to /config/fieldvalidators. This step also creates the /config/fieldvalidators/fieldvalidatorcountries directory for country-specific field validator overrides and ensures the proper schema location.

Replacing Field Validator Override and Field Length Override Elements

This step removes FieldValidatorOverride and FieldLengthOverride elements from the fieldvalidators.xml file.

The tool converts FieldValidatorOverride elements to column-override elements on the extension itself. For example, the default fieldvalidators.xml file in ClaimCenter 5.0 contained the following FieldValidatorOverride on the Contact object:

```
<FieldValidatorOverride
  entityname="Contact"
  fieldname="EmailAddress1"
  validatorname="Email"/>
```

The upgrade tool adds the following to the extensions/Contact.etx file:

```
<column-override name="EmailAddress1">
  <columnParam name="validator" value="Email"/>
</column-override>
```

The upgrade tool converts FieldLengthOverride elements to logicalSize elements on the extension.

Setting XML Namespace on Metadata Files

This step sets the XML namespace on data model and typelist entity and extension files in config/metadata and config/extensions to http://guidewire.com/datamodel and http://guidewire.com/typelists respectively. You can configure an XML editor to map these namespaces to XSD files that define the structure of data model and typelist files. Map http://guidewire.com/datamodel to ClaimCenter\modules\pl\xsd\metadata\datamodel.xsd and http://guidewire.com/typelists to ClaimCenter\modules\pl\xsd\metadata\typelists.xsd. Then, the XML editor can validate entities as you create or modify them.

Moving Typelist Localizations into Translation Properties Files

This step moves typelist localization elements into `translation.properties` files, in order to facilitate typelist merging. This step creates a `translation.properties` file for each locale containing the localized name and description of each localized typecode.

Upgrading PCF Files

This step upgrades PCF files from ClaimCenter 5.0.x format to ClaimCenter 6.0.8 format.

- The `afterCommit` attribute of popup page type PCF files is modified to replace `CurrentLocation` with `TopLocation`. `TopLocation` always represents the top location on a stack.
- `FinancialsSummaryLV.pcf` is updated as follows:
 - `FinancialsSummaryCell` is replaced with a `TextCell`. The `ColumnHeader` subelement value becomes a `label` attribute.
 - `FinancialsSummaryRow` is replaced with a `Row`.
 - `FinancialsSummaryLabel` name and value attributes are updated to match the new version of `FinancialsSummaryLV.pcf`, but otherwise stay the same.

The upgraded `FinancialsSummaryLV.pcf` does not exactly match the new version. Some simple merging is still required during the configuration merge step.

Upgrading Rules to Gosu Classes

This step upgrades rules to the new Gosu class format.

Rule Sets

The upgrader converts the old XML file format into a directory structure that represents the hierarchy of the rule set.

The top-level conversion converts the rule set into:

- a rule set Gosu class.
- a directory to contain all of the child rules. The directory is named using the convention `ruleset_dir`.

The rule set Gosu class contains the following upgraded rule set info:

- name (in the `RuleSetName` annotation)
- description
- type name (the Gosu class)

The directory contains the following:

- an `order.txt` file (representing the rules under this rule set)
- other rules for this class

The order of the top level rules (formerly in `ruleset.xml`) is now in `order.txt` under the rule set directory. The `order.txt` file contains relative type names for the rules to run. Individual rules can be disabled and that is represented by an annotation described below.

The rule set type is now encoded in the package name of the rule set. The code was changed to not use the type but instead use a readable package name.

The root bean object is encoded in the name of the type for the Preupdate and Validation rules.

Rules

Individual rules follow the same directory structure as rule sets. There is a primary Gosu class, a subdirectory (name of rule + `"_dir"`) and an `order.txt` file.

The rule set Gosu class contains the following upgraded data:

- rule set name (in the `RuleName` annotation)
- condition (converted into the `doCondition` method)
- action (converted into the `doAction` method)

The `messageContext` property used in the condition and action is converted as arguments to the `doCondition` and `doAction` methods. Furthermore the actions are converted into the `actions` parameter of the `doAction` method.

The action and condition are wrapped in the special tokens `/*start00rule/` and `/*end00rule*/`. These must be present or Studio cannot parse the file.

The order of subrules is now in the `order.txt` file along with relative type names for the rules.

Subrules under a rule follow the same process, recursively.

IMPORTANT You cannot perform a merge on rules because of this format change. Instead, after merging the rest of the configuration, use Studio to compare the ClaimCenter 6.0.8 default rules with your rules and update as necessary.

Adding Default Rules

This step moves the default rules provided with ClaimCenter 6.0.8 to a single folder. Because the configuration upgrade converts rules from XML to Gosu, you cannot perform a merge of rules as part of the upgrade. Instead, use Studio to compare the default rules with your custom rules and make updates.

Updating Plugin Template Suffixes

This step updates extension suffixes of plugin template files in `\config\templates\plugins` of each module from `.gs` to `.gsm`. Some older plugin interfaces use text generated by these templates to pass parameters from ClaimCenter to a Java or Gosu plugin implementation.

Removing AdminTable Delegate from Custom Extensions

This step removes the `AdminTable` delegate from custom extensions.

Upgrading ISO Coverage Mapping

This step produces mappings for each policy type mapped in `iso/TypeCodeMap.xml`. For each policy type, it finds all coverage types and coverage subtypes by using the categories in the `typelist` files as set up by the LOB editor. It then builds a mapping for each valid `policy type/coverage type/coverage subtype` triplet. If `TypeCodeMap.xml` does not contain a mapping for any one of the types (`policy`, `coverage` or `coverage subtype`), then the upgrader omits that triplet.

The upgrader finds applicable coverage types and coverage subtypes using case-insensitive matching. This is because `typelist` files are case-insensitive with respect to typecodes. However, the upgrader creates a new file, `ISOCoverageCodeMap.csv`, using the exact case given in the original `TypeCodeMap.xml`. For example, if typecode `abc` was sometimes listed as `ABC` in `typelist` categories but was listed as `ABC` in `TypeCodeMap.xml`, then it will appear as `ABC` in `ISOCoverageCodeMap.csv`.

The upgrader does not modify the `TypeCodeMap.xml` file, or any of the `typelist` files. If you will continue to use the old payload generation code, you will still need the original `TypeCodeMap.xml`.

This upgrade step is intended to produce a first draft of the `ISOCoverageCodeMap.csv` file. It does not take into account any custom configuration of the `policy type/coverage type/coverage subtype` that is done in the Gosu messaging code. Manually modify `ISOCoverageCodeMap.csv` to take account of such customizations.

Renaming Exchange Rate PCF Files

This step renames the two modes of `ExchangeRateInputSet.pcf` to be two separate files. These files are in the `config/web/pcf/claim/shared/` directory. The upgrader renames `ExchangeRateInputSet.Check.pcf` to `CheckExchangeRateInputSet.pcf` and `ExchangeRateInputSet.default.pcf` to `TransactionExchangeRateInputSet.pcf`.

Updating PCF Files to Reference Namespace and XML Schema

This step updates PCF files to reference the XML schema instance namespace and schema. The schema is provided with ClaimCenter 6.0 in `ClaimCenter\modules\pcf.xsd`. The configuration upgrade sets the schema location for each PCF file to point to the relative location of `pcf.xsd`. For example, the configuration upgrade adds the following to `ClaimCenter\modules\cc\config\web\pcf\admin\admin.pcf`:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../pcf.xsd">
```

Copying Display Properties Files into Target Configuration

This step copies `display.properties` files from the `locale` directories of the working configuration module to the target configuration module. The `display.properties` files do not require manual merging, so the Configuration Upgrade Tool copies them directly into the target configuration. If the file already exists in the target configuration, the tool skips the copy and logs a message. This is a precaution to make sure that the Configuration Upgrade Tool does not overwrite customized `display.properties` files if you run the tool again.

Reformatting Files

The Configuration Upgrade Tool performs a number of steps to reformat files so they are more easily merged during the manual merge process.

The tool sorts the content of `typelist` files in `/config/metadata` and `/config/extensions` by typecode to simplify the manual merge process.

The tool formats XML elements in files in the following directories to place tags on separate lines:

- `/config/metadata`
- `/config/extensions`
- `/config/web/pcf` and subfolders
- `/config/displaynames`
- `/config/workflow`
- `/config/locale/localization.xml`
- `/config/plugin/registry`

The tool removes carriage return characters from ASCII files, converting these files to UNIX format.

Specifying XML Namespaces

This step adds a namespace specification to certain XML files.

XML file	Namespace
<code>config/config.xml</code>	<code>http://guidewire.com/config</code>
<code>config/currency/currencies.xml</code>	<code>http://guidewire.com/currencies</code>
<code>config/locale/localization.xml</code>	<code>http://guidewire.com/localization</code>
<code>config/messaging/messaging-config.xml</code>	<code>http://guidewire.com/messaging-config</code>

Moving XSD Files to gsrc Directory

This step moves configuration module XSD files that are listed in `config/registry/registry-xsd.xml` from `configuration/config/registry/xsds` to `configuration/gsrc/xsd`. The Configuration Upgrade Tool uses the namespace listed for the XSD within `registry-xsd.xml` to create the new file name at the new location.

Copying Display Properties Files into Target Configuration

This step copies `display.properties` files from the `locale` directories of the working configuration module to the target configuration module. The `display.properties` files do not require manual merging, so the Configuration Upgrade Tool copies them directly into the target configuration. If the file already exists in the target configuration, the tool skips the copy and logs a message. This is a precaution to make sure that the Configuration Upgrade Tool does not overwrite customized `display.properties` files if you run the tool again. If you are upgrading from a version prior to 6.0, this step does not copy the file because an earlier automatic upgrade step already performed the copy.

Using the ClaimCenter 6.0.8 Upgrade Tool Interface

IMPORTANT Review the automated step descriptions before you proceed. Some automated steps might require you to perform a manual step while merging the configuration. Typically, such automated steps insert a warning into a file. Check the `steps_results.txt` file for warning and error messages. Correct any issues reported. Then, delete `steps_results.txt` and restart the Configuration Upgrade Tool.

After the Configuration Upgrade Tool completes the automated steps, the working area contains up to three versions of the same file:

- the *customer* file
- the *base* file, from which you configured the customer file
- the *target* file, from ClaimCenter 6.0.8.

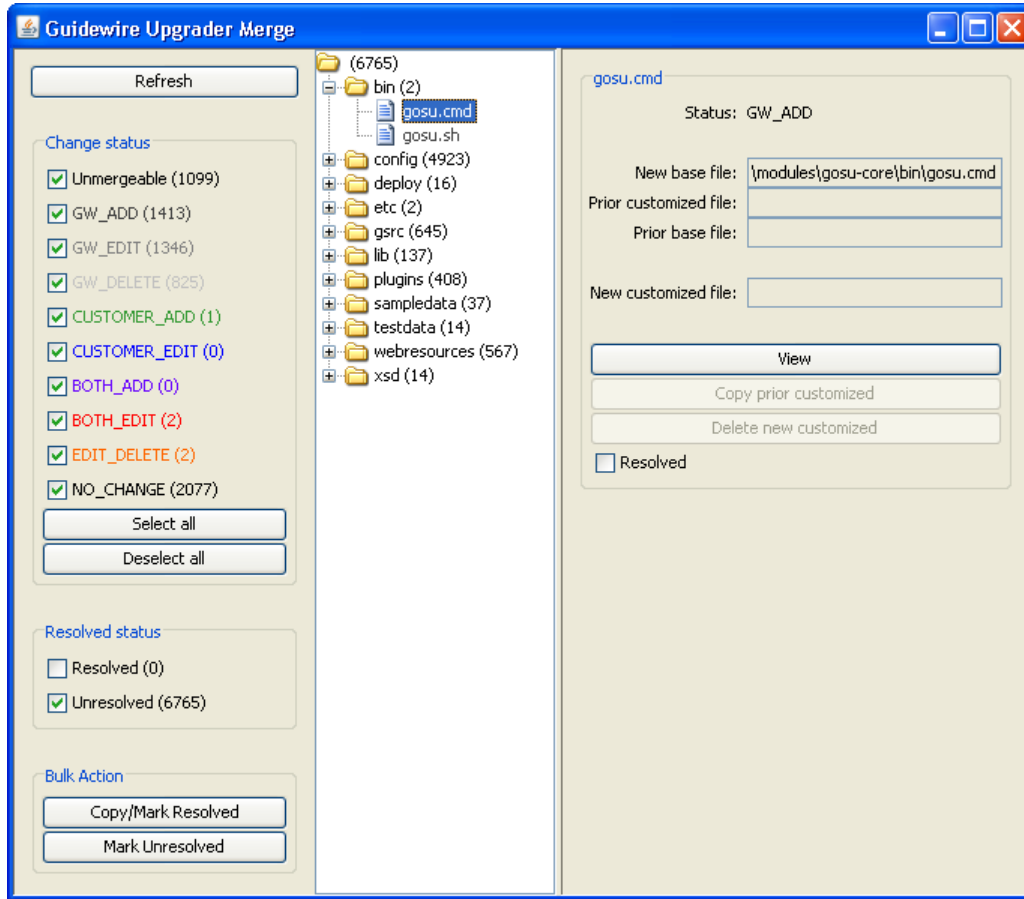
In the manual process of the upgrade, you decide whether to use one of these versions unchanged, or merge versions together. The Configuration Upgrade Tool provides a user interface to assist with the manual process. This interface has several important functions:

- It shows a complete list of all configuration files, organized into the module directory structure.
- It allows you to filter this list. You can, for example, view a list of all files that differ between the target version and your version. See “Change Status Filters” on page 93.
- It displays two or three versions of a file and their differences, using a merge tool you supply, such as Araxis Merge or P4Merge, defined in `upgrade.properties`.
- It lets you edit your file, incorporating changes from the other file versions, and save it.
- It lets you accept this merged version instead of one of the previous versions.
- It lets you edit the file after you have accepted changes from the merge using the text editor defined in `upgrade.properties`.

After you have accepted or merged all files that the Configuration Upgrade Tool displays, the merging process is complete.

The Configuration Upgrade Tool displays three panels. The center panel is a tree view of the files in the configuration, filtered by filter choices selected in the left panel. Files appear in the color of the filter that found them. As

you select a file in the center panel, the right panel displays file information and buttons to perform actions on that file.



Filters

The left panel of the Configuration Upgrade Tool contains:

- Refresh Button
- Change Status Filters
- Resolved Status Checkboxes
- Bulk Action Buttons

Refresh Button

If multiple users are working in the same directory, each user can mark files as resolved. The **Refresh** button refreshes the resolved status of files shown in the Configuration Upgrade Tool for changes contributed by all users working in the same directory.

Change Status Filters

This table lists the change status filters that the Configuration Upgrade Tool displays in the left panel. Use the check boxes next to the filters to select one or any combination of change statuses to view. Use the **Select all** or **Deselect all** buttons to select or deselect all filters. The following table describes change status filters. The

Guidewire Action column lists the change Guidewire has made to files matching a status filter since the prior version. The Your Action column lists the change to the file in your implementation:

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
Unmergeable	change format of file	any	file exists in a different format and thus cannot be merged with an old version	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file, in the new format, already exists in the target configuration.</p> <p>The Configuration Upgrade Tool automatically marks certain files as unmergeable, including rules and display properties files. The Configuration Upgrade Tool upgrades these files before the interface displays.</p> <p>You can also specify a regular expression pattern in <code>upgrade.properties</code> for file paths to mark files matching that pattern as unmergeable. Set the pattern as the value of the <code>exclude.pattern</code> property.</p> <p>Typically, you use <code>exclude.pattern</code> to specify source control metadata files. Samples are provided in <code>upgrade.properties</code> for CVS and SVN.</p>
GW_ADD	add	none	file in target not in base	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file added by Guidewire already exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
GW_EDIT	edit	none	file in target differs from base	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file added by Guidewire already exists in the target configuration.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a comparison between the new Guidewire version and the original base version. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
GW_DELETE	delete	none	file in base not in target	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file deleted by Guidewire no longer exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
CUSTOMER_ADD	none	add	file in customer configuration only	<p>If you resolve the file, the Configuration Upgrade Tool copies the file to the target configuration module if the file has not been copied there already.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
CUSTOMER_EDIT	none	edit	<p>file differs between customer and base configurations</p> <p>file unchanged between base and target configurations</p>	<p>If you resolve the file, the Configuration Upgrade Tool copies the file to the target configuration module if the file has not been copied there already.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a comparison between your custom version and the original base version. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
BOTH_ADD	add	add	new file with matching name in both target and customer configurations (rare)	<p>You must either merge the two versions of the file or copy your prior version of the file into the configuration module before you can resolve the file.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a merge between your version and the Guidewire version. If you make changes, the tool prompts you to copy the merged file to the target configuration module.</p>
BOTH_EDIT	edit	edit	file changed in both customer and target configurations	<p>You must either merge the two versions of the file or copy your prior version of the file into the configuration module before you can resolve the file.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.merge.tool</code> in <code>upgrade.properties</code> to perform a three-way merge between your custom version and the updated Guidewire version. If you make changes, the tool prompts you to copy the merged file to the target configuration module.</p>
EDIT_DELETE	delete	edit	file changed from base in customer configuration and does not exist in target configuration	<p>If you resolve the file, the Configuration Upgrade Tool takes no action.</p> <p>Double-clicking the file opens your customized file and the original base file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code>. When you close the merge tool, the Configuration Upgrade Tool prompts you to copy the file to the target configuration module. If you are sure you want your customized version, you can click Copy prior customized to move the file to the configuration module of the target version.</p> <p>The EDIT_DELETE flag appears on a file when your configuration has a customized version of the file but Guidewire has deleted the file from that location. There are two possible reasons for this deletion. One reason is that Guidewire removed the file from ClaimCenter. The second reason is that Guidewire has moved the file to a different folder.</p> <p>If Guidewire has completely removed the file, review the <i>ClaimCenter New and Changed Guide</i>, release notes, and the Upgrade Diffs report for descriptions of the change affecting the deleted file. Then determine if you want to continue moving your customization to the new or changed feature. If not, then the customization will be lost.</p> <p>For the second scenario, find where the file has been moved by searching the target version. Move your customized file to the same location in the working directory and make sure to match any case changes in the file-name. When you refresh the list of merge files, the file now appears under the CUSTOMER_EDIT filter. You can now proceed with the merge. If you do not move the file over, you can instead perform the merge manually by opening both files and incorporating the changes.</p>
NO_CHANGE	none	none	file not changed from base configuration in either customer or target configurations	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file already exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>

Resolved Status Checkboxes

Beneath the change status filters are checkboxes to toggle the visibility of resolved and unresolved files. Use these checkboxes with the change status filters to specify which types of files you want visible in the center panel. For example, you could select **BOTH_EDIT** and **Unresolved** to see files edited in your configuration that have also been updated by Guidewire and are not yet resolved.

The purpose of the resolved status is to have a general idea of the progress you are making in the upgrade. The tool shows the resolved status of the current file (right panel) and the total number of resolved and unresolved files (left panel).

A resolved file is simply a file that you have marked resolved. It does not relate to whether file merging or accepting has occurred.

Bulk Action Buttons

The following buttons in The **Bulk Action** part of the left panel enable you to change the resolved status of a group of selected files:

- **Copy/Mark Resolved**
- **Mark Unresolved**

You can select either one or several files and directories before using these buttons. Use the CTRL key to select multiple files and directories. Selecting a directory selects all files within that directory. You can select all files that match the filters you set by selecting the top-level directory.

After you click **Copy/Mark Resolved**, the Configuration Upgrade Tool opens a dialog detailing the actions it is about to perform.

The tool copies files matching the **CUSTOMER_ADD** and **CUSTOMER_EDIT** filters to the target configuration module. If there is already a version of a file in the target configuration module, then the tool does not copy the file. A file would be there already if you edited the file and clicked **Yes** when the tool prompted you to copy the file to the configuration module.

The tool does not do any copying for files matching the **GW_ADD**, **GW_DELETE**, **GW_EDIT**, **NO_CHANGE**, or **Unmergeable** filters. Files matching **GW_ADD**, **GW_EDIT**, **NO_CHANGE**, or **Unmergeable** filters are already present in the target version. Files matching the **GW_DELETE** filter are not in ClaimCenter 6.0.8.

You can not bulk resolve multiple files that match the **BOTH_ADD**, **BOTH_EDIT**, or **EDIT_DELETE** filters. Files matching these filters require individual attention. Use the right panel of the Configuration Upgrade Tool to control merging, copying and resolving of these files.

Configuration File Tree

The center panel displays the configuration file tree. Files are color-coded to match filter colors. Files are shown one time, regardless of the number of configurations in which they exist. For information on which configurations a file exists in, select the file and view the right panel. The number of files in each directory that match the selected change status and resolved status filters is shown in parentheses.

File Details Panel

The right panel displays file details for the file you are currently examining, including:

- **Status:** the change status of the file. See “Change Status Filters” on page 93.
- **New base file:** The new version of this file supplied by Guidewire, typically in either the cc or p1 module of ClaimCenter 6.0.8. If there is not a Guidewire version of this file, such as for **CUSTOMER_ADD**, **EDIT_DELETE** or **GW_DELETE** files, this field is blank.
- **Prior customized file:** The locally customized version of this file from the prior version. If there is not a customized version of this file, such as for **GW_ADD**, **GW_DELETE**, **GW_EDIT** or **NO_CHANGE** files, this field is blank.

- **Prior base file:** The base version of this file in the working directory. If there is not a Guidewire version of this file in the prior base version you are upgrading from, such as for **CUSTOMER_ADD** files, this field is blank.
- **New customized file:** The customized version of this file in the ClaimCenter 6.0.8 configuration directory.

The right panel fields are blank if you have multiple files selected.

File Details Panel Actions

The following buttons appear below the file details display in the right panel after you have selected a file:

- **View:** Opens the file in the editor specified in `upgrade.properties`. This button appears for files that are not customized and do not require merging, matching **GW_ADD**, **GW_EDIT**, or **GW_DELETE** filters. Only one of the **View**, **Edit** or **Merge** buttons displays, depending on the file change status.
- **Edit:** Opens the file in the editor specified in `upgrade.properties`. This button appears for custom files that do not require merging, matching the **CUSTOMER_ADD** or **EDIT_DELETE** filters.
- **Merge:** Opens the different versions of the file in the merge tool specified in `upgrade.properties`. This button appears for files that require merging, matching the **BOTH_ADD** or **BOTH_EDIT** filters.
- **Copy prior customized:** Copies the prior customized version of the file to the target configuration module. This button is enabled if there is a prior customized version of the file. So it is enabled for files matching **CUSTOMER_ADD**, **CUSTOMER_EDIT**, **BOTH_ADD**, **BOTH_EDIT**, or **EDIT_DELETE** filters.
- **Delete new customized:** Remove the customized version from the configuration module. This reverses the **Copy prior customized** button action. This button is disabled until you have copied a prior customized version of the file into the configuration module.
- **Resolved:** Check this box to label the file resolved. Use the **Resolved** checkbox in the right pane to change the status of a single file. Selecting the **Resolved** checkbox does not copy the file. Use the buttons above this checkbox to handle copying or merging of file versions. You must first unresolve a file before either using the **Delete new customized** action or reapplying changes or merges.

Accepting Files that Do Not Require Merging

The following filters show lists of files that normally do not require merging.

- **CUSTOMER_ADD**
- **CUSTOMER_EDIT**
- **GW_ADD**
- **GW_EDIT**
- **NO_CHANGE**
- **Unmergeable**

You can click the **Copy/Mark Resolved** button in the left panel to resolve groups of these files.

Merging and Accepting Files

Files matching the **BOTH_ADD** and **BOTH_EDIT** filters must be merged before being accepted. Your version must be reconciled with the Guidewire target or base version. In some cases, even if only a single version of the file exists, you might want to look at it before accepting it.

You can use the `pcf.xsd` file in `modules/cc/config/web/schema` of the target version to validate merged PCF files.

After you are satisfied with any changes, save the file. This saves the file in a temporary directory. When you close the editor or merge tool, the Configuration Upgrade Tool asks if you want to copy the file to the target configuration module. If you click **Yes**, the tool copies the file. If you click **No** (or **CTRL+N**), the tool cancels the popup without copying. The tool always moves files into the configuration module, except if a file is identical to the base or target version. In this case, the tool does not move the file.

Note: Do not edit a file version with **DO_NOT_EDIT** in its file name.

Configuration Merging Guidelines

First, work to generate the toolkit. The first milestone of an upgrade project is to generate a toolkit (by running `gwcc regen-java-api` and `gwcc regen-soap-api`) on the target release. To do this, you must:

- Complete the merge of the data model. This includes all files in the `/extensions` and `/fieldvalidators` folders.
- Resolve issues encountered while trying to generate the toolkit or start the QuickStart application server.

You can generate a toolkit even if you have errors in your enhancements, rules and PCF files.

Typical errors

- **Malformed XML:** The merge tool is not XML-aware. There might be occasions in which the file produced contains malformed XML. To check for well-formed XML, use free third-party tools such as Liquid XML, XML Marker, or Eclipse.
- **Duplicate typecodes:** As part of the merge process, you might have inadvertently merged in duplicate, matching typecodes.
- **Missing symmetric relationship on line-of-business-related typelists:** You might be missing a parent-child relationship with respect to the line-of-business-related typelist, as a result of merging.
- **References to GScript instead of Gosu,** such as in `plugins/registry`.

Once the toolkit has been generated, you can begin the work of upgrading integrations. See “Upgrading Integrations and Gosu from 5.0.x” on page 167.

Second, after you can successfully generate the Java and SOAP APIs, work on starting the server.

In addition to the typical issues above, the server might fail to start due to cyclical graph reference errors. See “Identifying Data Model Issues” on page 120.

You can generate a toolkit even if you have errors in your enhancements, rules and PCF files, although the error messages will print upon server startup.

Once the server can start on the target release, you can begin the database upgrade process.

Continue with the remainder of the configuration upgrade work, which includes evaluating existing PCF files and merging in desired changes.

Data Model Merging Guidelines

From a purely technical standpoint, not addressing the need to incorporate new features, the following are a few guidelines for merging the data model.

Merging Typelists: Overview

There is no automated process to merge typelists. This is a part of the merge process using the Configuration Upgrade Tool. In general, merge typelists before PCF files.

Merge in Guidewire-provided typecodes related to lines of business and retire unused typecodes that you merge in. If you do not include these typecodes, you will have errors in any enhancements, rules, or PCF files that reference the typecode. This also simplifies the process for future upgrades as there will be fewer added line of business typecodes to review.

Pay particular attention if any Guidewire-provided typecodes have the same typecode as a custom version. In this case, modify one of the typecodes so they are unique. Contact Guidewire Support for details.

The Configuration Upgrade Tool displays most typelists you have edited in the `CUSTOMER_EDIT` filter. If your edits are simply additional typecodes, accept your version.

Use Guidewire Studio to verify PCF files, enhancements, and rules to identify any issues with the files and rules that reference typelists.

Merging Typelists: Simple Typelists

Merge in new typecodes from the target version, ClaimCenter 6.0.8. If you do not merge the new typecode, you will have errors in any enhancements, rules, or PCF files that reference the typecode. If you do not want to use a new typecode, retire the typecode by setting the `retired` attribute to `true`.

In the following example, the target version of ClaimCenter (shown on left) introduced the `debrisremoval` typecode.



To avoid errors in enhancements, rules, and PCF files that reference the `debrisremoval` typecode, merge the typecode. If you do not want to use the new typecode, merge it into the target file and retire it by setting the `retired` attribute to `true`, as shown below.



Merging Typelists: Complex Typelists

A typecode can reference typecode values from another typelist using the `<category>` subelement. If a new typecode references an existing typecode, do not merge the new typecode unless the referenced typecode is retired. Otherwise, you are defining a new relationship. If the referenced typecode is also new, merge in both typecodes. If you do not want to use a new typecode, set the `retired` attribute for the typecode to `true`. The following table summarizes how to handle merging new typecodes that reference other typecodes:

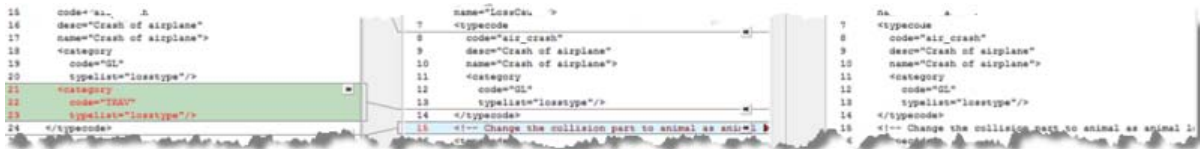
Referenced typecode status	Action
new: exists only in target version	Merge in the new typecode and merge in the referenced typecode in its typelist. If you do not want to use the new typecode, retire it by setting the <code>retired</code> attribute of the typecode to <code>true</code> .
active: exists in base or custom version and is not retired	Do not merge the new typecode.
retired: exists in base or custom version and is retired	Merge in the new typecode. If you do not want to use the new typecode, retire it by setting the <code>retired</code> attribute of the typecode to <code>true</code> .

In the following example, the typecode `abandonment` did not exist previously, either in the old default configuration or in the custom configuration. If you do not want the new typecode, Guidewire recommends that you merge and retire the typecode and all its children. This assumes that you are following other recommendations such that the `TRAVLOS` type referenced is also merged in, and retired if applicable. Because this typecode is retired, there

are no harmful consequences of this action. If you choose to unretire the typecode in the future, you will have the desired, default behavior in place.



In the next example, `air_crash` is a typecode that is already existing and active in the original base version. The latest version introduces a new `<category>` `losstype` subelement of `TRAV` to the `air_crash` typecode. If `TRAV` is already an active typecode, do not merge in the new `<category>` subelement. If you do, you will create a line of business relationship in the data model which did not previously exist. Merging in category subelements that reference active typecodes might produce unwanted consequences.



Note: The `LineCategory` typelist tends to be very large. Guidewire recommends that you select the **Copy Prior Customized** option and not merge the `LineCategory` typelist. If you are keeping your existing line of business configuration, your customized `LineCategory` typelist is sufficient for the upgrade.

Merging Entities

Guidewire recommends that you merge in newly defined indexes from the target configuration.

For example, in the `Claim.etx` file, merge in the following changes to your customized configuration:

```
<index
  desc="Covering index for helping to speed up Claim Search by Insured's last name, when including
    losdate as one of the criteria"
  expectedtobecovering="true"
  name="claimu6"
  trackUsage="true"
  unique="true">
  <indexcol
    keyposition="1"
    name="InsuredDenormID"/>
  <indexcol
    keyposition="2"
    name="Retired"/>
  <indexcol
    keyposition="3"
    name="LossDate"/>
  <indexcol
    keyposition="4"
    name="ID"/>
</index>
```

Merging Contact Entities That Have Foreign Key or Array Extensions

If you have extended the `Contact` entity or any `Contact` subtentities by adding foreign keys or arrays, you must edit each added entity and have it implement `AddressBookLinkable`. Additionally, if you have extended `ABContact` or any of its subtentities in `ContactCenter` in the same way, these added entities must each implement `ABLinkable`.

For example:

- If you extended the `Contact` entity in `ClaimCenter` with an array or foreign key, you would add the following element to the `.etx` file for the new entity. Add the element to the entity that populates the array or the entity to which the foreign key links:

```
<implementsEntity name="AddressBookLinkable"/>
```

- If you extended the ABContact entity in ContactCenter with an array or foreign key, you would add the following element to the .etx file for the new entity. Add the element to the entity that populates the array or the entity to which the foreign key links:

```
<implementsEntity name="ABLinkable"/>
```

Merging Other Files

In some cases, the differences between files cannot be merged effectively using a comparison tool. In particular, `config.xml`, `logging.properties`, and `scheduler-config.xml` often have many changes between major versions. Consider adding your custom changes to the new Guidewire-provided version instead of merging from prior versions if the presentation of these files in the merge tool is too daunting.

IMPORTANT ClaimCenter 6.0.8 includes a `StrictDataTypes` parameter in `config.xml`. This parameter controls validation of data by ClaimCenter. The default value for `StrictDataTypes` is `true`. Guidewire recommends that you set `StrictDataTypes` to `false` to perform the upgrade. Setting `StrictDataTypes` to `false` preserves the existing data validation behavior. For more information about the `StrictDataTypes` parameter, see “`StrictDataTypes`” on page 55 in the *Configuration Guide*.

Upgrading Rules

The Configuration Upgrade Tool upgrades rules to Gosu classes instead of the former XML format. You cannot perform a merge on rules because of this format change. Instead, after merging the rest of the configuration, use Studio to compare the ClaimCenter 6.0.8 default rules with your rules and update as necessary.

See “Upgrading Rules to ClaimCenter 6.0.8” on page 108.

About Display Properties

You do not need to manually merge `display.properties`. ClaimCenter collects display properties from both the `cc` and `configuration` modules. This is different than the way ClaimCenter handles most other files. For most files, if a version exists in the `configuration` module, ClaimCenter uses that version instead of the version in the `cc` module. For `display.properties`, ClaimCenter creates a union of values from both files.

During the upgrade, any custom display properties are moved to the `configuration` module. Any display properties that have been added by Guidewire are in the `cc` module. Although `display.properties` appears in the Configuration Upgrade Tool interface, you do not need to merge this file. ClaimCenter essentially performs the merge for you each time it starts.

If you have added locales, you can export a full list of display keys and typelists from the default ClaimCenter 6.0.8 locale to any locale you have defined. This list includes a section for display keys and typelists that do not yet have values defined for your locale. You can use this list to determine which display keys and typelists require localized values. You can then specify those values and import the list.

See “Translating New Display Properties and Typecodes” on page 110.

Method Removed from Typekeys

In ClaimCenter 5.0.4, Guidewire removed the `valueOf` method that was present on all typekeys. If you are upgrading from a version prior to 5.0.4, replace calls to the `valueOf` method with calls to the `getByCode` method.

Merging PCF Files

This topic provides some guidance for merging PCF file changes for specific features. Review the PCF file changes described in the automated steps for changes that apply to all PCF files.

General PCF Changes

You can no longer create a PCF page using a Search Panel widget as the top-level element. You can, however, still use this widget as a lower-level element. None of the PCF pages provided with previous versions of ClaimCenter used a Search Panel widget as the top-level element. However, you might have created such a PCF page in your custom configuration. If you have, modify the PCF page so that the Search Panel widget is not the top-level element.

Address Consolidation

ClaimCenter had multiple views to display addresses for different formats and entities. For example, the view for `Claim.LossLocation` was different from `Exposure.TempLocation`. Both of these are addresses, but the views used were different.

Different address views posed a challenge for maintenance and localization. For example, if you did not want to display **Address Line 2**, you needed to configure multiple views. Similarly, if you needed to add an international address format for another country, that too required you to configure multiple files.

To resolve this issue, ClaimCenter 6.0 includes a single address view that can have modes for each country code. This provides one view that you can customize for the examples stated above.

The new `AddressOwner` interface specifies which entity owns a particular address, so that the address can be saved properly. For example, if the address is a loss location, it must be saved on `Claim`. If it is a temporary location for loss of use, then it must be saved on `Exposure`. All entities with addresses extend this interface. Custom extensions are supported.

AddressOwner and CCAddressOwner

The `AddressOwner` is the interface to create a helper object that is passed to the `AddressInputSet`. The helper object provides a way to set and get a single address on the enclosing entity. It also provides methods to allow fields to be required, visible, and so forth.

The `CCAddressOwner` interface extends the `AddressOwner` interface to add ClaimCenter specific fields. A `CCAddressOwner` object is passed as the argument to the generic `AddressInputSet`. The generic `AddressInputSet` calls methods and properties on `CCAddressOwner` to determine which fields to show, which fields are required, how to get and set the `Address` object, and so forth. This allows a single modal `AddressInputSet` to be reused in many different situations, making it easier to add new countries to the system.

Overall, there are different levels at which you can configure address fields:

- Simple global changes can be made by altering the constant sets of values in `CCAddressOwnerFieldId`.
- Country-specific changes can be made by adding a country-specific `CountryAddressFields` object. Refer to the `CountryAddressFields` class for details.
- Changes specific to a particular use of `AddressInputSet` can be made by adding or altering the particular `CCAddressOwner` passed into that input set. This owner object can override any country-specific or global defaults.

New Address User Interface

The new user interface is intended to be consistent across ClaimCenter with a few exceptions as noted.

In view mode, ClaimCenter always displays the address on two lines. This was done by adding a new line to the `Address` entity name. The description field is now available for all addresses.

In edit mode, the address section looks similar to earlier versions of ClaimCenter.

The address user interface always displays a range selector to allow selection from an appropriate address array. If the address is not editable, the fields are grayed out. The range selector can display **New...** to allow creation of a new address.

Address PCF Configuration

The shared address views are located under `/cc/config/web/pcf/claim/shared/address`.

Each `AddressInputSet.XX.pcf` relates to a locale and is referenced using an `InputSetRef` widget. For instance, from `LossDetailsDV.Auto.pcf`:

```
<InputSetRef
  def="AddressInputSet(Claim.AddressOwner)"
  mode="Claim.AddressOwner.InputSetMode" />
```

In the example, the `Claim.AddressOwner` parameter passed to `AddressInputSet` is a `CCAddressOwner` object. The mode is the switch to select the correct view mode. If a country selected from the drop-down does not exist as a mode, ClaimCenter uses `AddressInputSet.default.pcf`.

Gosu Configuration for a New Address Relationship

You might need to delegate an entity as an `AddressOwner` when you add an `Address` foreign key to:

- an existing entity
- a new entity

For both cases:

1. First create a new Gosu class that implements the `AddressOwner` or `CCAddressOwner` interface. Studio prompts you to implement the methods.
2. Press ALT + ENTER to implement the methods. See `VehicleIncidentAddressOwner.gs` and `ClaimRelatedAddressOwner.gs` for an implementation example.
3. Create a new Gosu enhancement to enhance the new or existing entity with `get` and `set` methods. See `GWTripAccommodationAddressOwnerEnhancement.gsx` for sample implementation details.

Address Files Removed

The following redundant files were removed as a part of the address consolidation:

- `LossDetailsAddressDetailInputSet.default.pcf`
- `LossDetailsAddressDetailInputSet.CA.pcf`
- `LocationDetailInputSet.CA.pcf`
- `LocationDetailInputSet.US.pcf`
- `LocationDetailInputSet.default.pcf`
- `LossOfUseTempAddressInputSet.CA.pcf`
- `LossOfUseTempAddressInputSet.US.pcf`
- `LossOfUseTempAddressInputSet.default.pcf`
- `AddressBookContactAddressDetailInputSet.CA.pcf`
- `AddressBookContactAddressDetailInputSet.US.pcf`
- `AddressBookContactAddressDetailInputSet.default.pcf`
- `UserAddressDetailInputSet.CA.pcf`
- `UserAddressDetailInputSet.US.pcf`
- `UserAddressDetailInputSet.default.pcf`
- `ContactAddressDetailInputSet.CA.pcf`
- `ContactAddressDetailInputSet.US.pcf`
- `ContactAddressDetailInputSet.default.pcf`
- `FixedPropertyIncidentDV.CA.pcf`
- `FixedPropertyIncidentDV.default.pcf`
- `AddressDetailDV.CA.pcf`
- `AddressDetailDV.US.pcf`
- `AddressDetailDV.default.pcf`

Multiple inline addresses screens have been updated to use the `AddressInputSet` to further ease address configuration across the system.

ContactCenter Address Input

ContactCenter 6.0.8 no longer includes the following address input PCF files: `AddressDetailInputDVSet.pcf` and `AddressAutofillableInputSet.pcf`. If you have custom PCF files that reference these files, update the references to instead use `AddressDetailInputSet.pcf` or `AddressAutofillableDVInputSet.pcf`, respectively. If you customized `AddressDetailInputDVSet.pcf` or `AddressAutofillableInputSet.pcf`, review `AddressDetailInputSet.pcf` and `AddressAutofillableDVInputSet.pcf`, respectively and apply any required customizations.

Homeowners FNOL

This section lists the files that Guidewire has added, changed or removed as part of the change to the homeowners FNOL. You might choose not to use the new homeowners FNOL because you already have customized FNOL files for a homeowners line of business.

All file names are given from the base of `/cc/config/web/pcf`.

Files Added

All the `/claim/snapshot/600` pages are new. These files include changes to display homeowner data. These snapshot PCF files are not included in the list below.

Other files added for homeowners FNOL include:

- `/claim/FNOL/FNOLContactInputSet.pcf`
- `/claim/FNOL/FNOLInjuryIncidentPopup.pcf`
- `/claim/FNOL/FNOLWizard_BasicInfoPolicyPanelSet.Homeowners.pcf`
- `/claim/FNOL/FNOLWizard_BasicInfoRightPanelSet.Pr.pcf`
- `/claim/FNOL/FNOLWizard_NewLossDetailsPanelSet.homeowners.pcf`
- `/claim/FNOL/FNOLWizard_NewLossDetailsScreen.PR.pcf`
- `/claim/FNOL/FNOLWizard_ServicesPolicyPanelSet.Homeowners.pcf`
- `/claim/FNOL/FNOLWizard_ServicesScreen.Auto.pcf`
- `/claim/FNOL/FNOLWizard_ServicesScreen.Pr.pcf`
- `/claim/FNOL/FixedPropertyIncidentDebrisRemovalInputSet.pcf`
- `/claim/FNOL/FixedPropertyIncidentEMSInputSet.pcf`
- `/claim/FNOL/NewClaimContentsDamageDV.pcf`
- `/claim/FNOL/NewClaimExposureDV.Content.pcf`
- `/claim/FNOL/NewClaimExposureDV.LivingExpenses.pcf`
- `/claim/FNOL/NewClaimExposureDV.OtherStructure.pcf`
- `/claim/FNOL/NewClaimLivingExpensesDV.pcf`
- `/claim/FNOL/NewClaimOtherStructureDamageDV.pcf`
- `/claim/exposures/ContentsDamageDV.pcf`
- `/claim/exposures/ExposureDetailDV.Content.pcf`
- `/claim/exposures/ExposureDetailDV.LivingExpenses.pcf`
- `/claim/exposures/ExposureDetailDV.OtherStructure.pcf`
- `/claim/exposures/ExposureDetailPopup.pcf`
- `/claim/exposures/LivingExpensesDV.pcf`
- `/claim/exposures/OtherStructureDamageDV.pcf`
- `/claim/lossdetails/EditableRoomsLV.pcf`
- `/claim/lossdetails/FireDamageInfoInputSet.pcf`
- `/claim/lossdetails/FireDamageQuestionsPanelSet.pcf`
- `/claim/lossdetails/LossDetailsPanelSet.Pr.pcf`
- `/claim/lossdetails/LossDetailsPanelSet.pcf`
- `/claim/lossdetails/LossDetailsPrPanelSet.homeowners.pcf`
- `/claim/lossdetails/LossDetailsPrPanelSet.pcf`
- `/claim/lossdetails/RelatedExposuresLV.pcf`
- `/claim/lossdetails/WaterDamageQuestionsPanelSet.pcf`

- /claim/lossdetails/print/DwellingIncidentPrint.pcf
- /claim/lossdetails/print/InjuryIncidentDetailPrint.pcf
- /claim/lossdetails/print/InjuryIncidentsPrint.pcf
- /claim/lossdetails/print/LivingExpensesIncidentPrint.pcf
- /claim/lossdetails/print/OtherStructureIncidentPrint.pcf
- /claim/lossdetails/print/PropertyContentsIncidentPrint.pcf
- /claim/newexposure/NewExposureDV.Content.pcf
- /claim/newexposure/NewExposureDV.LivingExpenses.pcf
- /claim/newexposure/NewExposureDV.OtherStructure.pcf
- /claim/planofaction/ClaimEvaluationDetailsDV.Trav.pcf
- /claim/shared/Property/PropertyAddressInputSet.CA.pcf
- /claim/shared/Property/PropertyAddressInputSet.US.pcf
- /claim/shared/Property/PropertyAddressInputSet.default.pcf

Files Edited

- /admin/reinstthreshold/ReinsuranceThresholdLV.pcf
- /claim/FNOL/FNOLContactPopup.pcf
- /claim/FNOL/FNOLWizard.pcf
- /claim/FNOL/FNOLWizardFindPolicyPanelSet.Search.pcf
- /claim/FNOL/FNOLWizard_BasicInfoRightPanelSet.Auto.pcf
- /claim/FNOL/NewClaimExposureDV.Propertydamage.pcf
- /claim/FNOL/NewClaimLossDetailsDV.Auto.pcf
- /claim/FNOL/NewClaimLossDetailsDV.GL.pcf
- /claim/FNOL/NewClaimLossDetailsDV.Pr.pcf
- /claim/FNOL/NewClaimLossDetailsDV.Wc.pcf
- /claim/lossdetails/ClaimLossDetails.pcf
- /claim/lossdetails/DwellingIncidentPanelSet.pcf
- /claim/lossdetails/LossDetailsDV.GL.pcf
- /claim/lossdetails/LossDetailsDV.Pr.pcf
- /claim/lossdetails/LossDetailsPanelSet.Homeowners.pcf
- /claim/lossdetails/OtherStructureIncidentPanelSet.pcf
- /claim/lossdetails/PrDV.pcf
- /claim/lossdetails/PropertyAttributeInputSet.pcf
- /claim/lossdetails/PropertyContentsIncidentPanelSet.pcf
- /claim/medicalcasemgmt/ClaimWCmedCaseMgmt.pcf
- /claim/summary/indicator/ClaimIndicatorInputSet.FlagClaimIndicator.pcf
- /shared/printing/ClaimPrintout.pcf

Files Deleted

- /claim/FNOL/FNOLWizard_ServicesScreen.pcf
- /claim/lossdetails/FixedPropertyIncidentDV.CA.pcf
- /claim/lossdetails/FixedPropertyIncidentDV.default.pcf

Fixed Property Incident Detail View

In FixedPropertyIncidentDV.pcf the LienHoldersLV list view is set to be visible only if the PolicyType is not homeowners. To see the data for homeowners policies remove the visible attribute from the ListViewInput parameter.

```
<ListViewInput
def="EditablePropertyLienholdersLV(FixedPropertyIncident.Property, Claim)"
editable="Claim.canEditProperty(FixedPropertyIncident.Property)"
label="displaykey.Web.FixedPropertyIncident.Property.Lienholders"
validationExpression="FixedPropertyIncident.Property != null
FixedPropertyIncident.Property.validateLienholders() : null"
visible="FixedPropertyIncident.Claim.Policy.PolicyType != &quot;homeowners&quot;">
```

Incident Type-specific Behavior

ClaimCenter adds new exposure and incident types and maps these exposure and incident types to existing coverage subtypes. See “Remapping Incidents” on page 152. You can modify this mapping by changing the mapping of your coverage subtypes to exposure types.

The updated line of business configuration will result in incident subtypes changing. Be prepared for any type-specific behavior of the new incident type. For the new Dwelling incident type, one such behavior is that the loss location of the claim is used as the location for the incident. Any location explicitly set on an incident that becomes a Dwelling incident will not be visible in ClaimCenter.

By default, the ClaimCenter upgrade remaps Fixed Property Incidents with the following coverage subtypes to Dwelling incidents:

- Dwelling - PropertyDamage
- Earthquake - Dwelling - PropertyDamage
- Flood - Dwelling - PropertyDamage
- Mold - Dwelling - PropertyDamage

PCF File Changes Required if not Remapping Property Exposures

ClaimCenter remaps Property exposures mapped to the Fixed Property Incident type to Dwelling or Other Structure incident types, depending on the coverage subtype. See “Remapping Incidents” on page 152. You can modify this mapping before you run the database upgrade. If you change this mapping so that these Property exposures stay mapped to the Fixed Property Incident type, update PCF files that reference `Exposure.DwellingIncident` or `Exposure.OtherStructureIncident` to instead reference `Exposure.FixedPropertyIncident`.

The following PCF files reference `Exposure.DwellingIncident`:

- `config/web/pcf/claim/exposures/PropertyIncidentInputSet.Dwelling`
- `config/web/pcf/claim/FNOL/NewClaimIncidentInputSet.Dwelling`
- `config/web/pcf/claim/snapshot/600/ClaimSnapshotExposure600DV.Dwelling`

The following PCF files reference `Exposure.OtherStructureIncident`:

- `config/web/pcf/claim/exposures/OtherStructureDamageDV`
- `config/web/pcf/claim/exposures/NewClaimOtherStructureDamageDV`
- `config/web/pcf/claim/snapshot/600/ClaimSnapshotExposure600DV.OtherStructure`

Holidays

ClaimCenter 6.0 adds `ZoneType` and `Country` to `HolidayZone` entities. These fields help to more precisely define a `HolidayZone`. As businesses expand into more territories, this becomes necessary to retain unique `HolidayZones`. During holiday configuration, ClaimCenter 6.0 enables you to specify the zone type and country to which the holiday apply. You cannot use the holiday configuration PCF files from a prior version. Accept the new Guidewire versions of files in `config/web/pcf/admin/holidays`. If you want to make changes to any of these files, you can customize the new versions.

Regions

ClaimCenter 6.0 removes the deprecated `Region.RegionType` field. If you have customized region configuration PCF files that used this field, you must remove it.

ClaimCenter 5.0 PCF files did not include `Region.RegionType` by default.

ClaimCenter 4.0 included `Region.RegionType` by default in the following PCF files:

- `config/web/pcf/admin/regions/RegionDetailDV`
- `config/web/pcf/admin/regions/RegionsLV`

Catastrophes

ClaimCenter 6.0 changed from using `CatastropheState` entities to `CatastropheZone` entities. However, you can preserve `CatastropheState` data by creating an extension on `Catastrophe` prior to upgrading the database.

Guidewire has updated PCF files that reference `CatastropheState` and has removed the `config/web/pcf/admin/catastrophes/CatastropheStateLV` file. If you keep the `CatastropheState` entity, look for PCF files that have changed from referencing `CatastropheState` to `CatastropheZone` and include the `CatastropheState` references in the PCF files. In particular, Guidewire has removed `CatastropheState` from `config/web/pcf/admin/catastrophes/AdminCatastropheDV`.

You might have decided to switch from `CatastropheState` entities to `CatastropheZone` entities and decided to preserve `CatastropheState` comments. If you created the `Comments` column extension on `CatastropheZone`, you might want to add `CatastropheZone.Comments` to `catastrophe` PCF pages.

Deductible Fields

ClaimCenter 6.0 adds a new `Deductible` entity. The database upgrade removes the following deductible fields from `Claim`, `Transaction` and `Payment` if they are empty.

- `Claim.DeductibleStatus`
- `Transaction.DeductiblePaid`
- `Payment.DeductibleAmount`
- `Payment.DeductibleSubtracted`
- `Payment.NetAmount`
- `Payment.OriginalAmount`

However, if you have data in these fields that you want to keep, you can create extensions to preserve these fields in the database.

Guidewire has updated PCF files that reference these deductible fields. If you preserve the fields, keep the references to the deductible fields when merging PCF files.

Prior versions of ClaimCenter included `Claim.DeductibleStatus` on the following PCF files by default:

- `config/web/pcf/claim/lossdetails/LossDetailsDV.Auto`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.GI`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Pr`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Wc`

These references have been removed in ClaimCenter 6.0. None of the other deductible fields listed were referenced on ClaimCenter PCF files by default.

Claim snapshot PCF files show a claim as it was originally created. Claim snapshots are stored as XML rather than in the `cc_claim` table, so they are not affected by later changes to the `Claim` entity. So some claim snapshot PCF files include the `DeductibleStatus` because it was previously a property on `Claim`. These files include:

- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Auto`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.GI`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Pr`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Wc`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Auto`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.GI`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Pr`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Wc`

For more information on claim snapshots, see “How ClaimCenter Renders Claim Snapshots” on page 629 in the *Configuration Guide*.

Reinsurance Status

Guidewire has updated PCF files to remove references to `Claim.ReinsuranceStatus`. If you keep this field by creating an extension, look for PCF files that have `Claim.ReinsuranceStatus` references removed and add the reference.

Prior versions of ClaimCenter included `Claim.ReinsuranceStatus` in the following files by default:

- `config/web/pcf/claim/lossdetails/LossDetailsDV.Auto`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Gl`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Pr`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Wc`

These references have been removed in ClaimCenter 6.0.

Claim snapshot PCF files show a claim as it was originally created. Claim snapshots are stored as XML rather than in the `cc_claim` table, so they are not affected by later changes to the `Claim` entity. So some claim snapshot PCF files include the `ReinsuranceStatus` because it was previously a property on `Claim`. These files include:

- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Auto`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Gl`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Pr`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Wc`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Auto`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Gl`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Pr`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Wc`

For more information on claim snapshots, see “How ClaimCenter Renders Claim Snapshots” on page 629 in the *Configuration Guide*.

Merging Document, Email and Note Templates

Guidewire has updated templates in `config/resources/` for document, email, and note production to support localized versions. Guidewire has deprecated a number of template methods. These methods continue to work in ClaimCenter 6.0.8, but will be removed in a future release.

Upgrading Rules to ClaimCenter 6.0.8

The Configuration Upgrade Tool does not upgrade rules. The tool classifies rules in the unmergeable filter. Within the target directory, Guidewire-provided default rules are located in `modules/cc/config/rules` and `modules/pl/config/rules`. The Configuration Upgrade Tool moves your custom rules to `modules/configuration/config/rules`.

Guidewire also copies the default rules for the current release into a ClaimCenter 6.0.8 folder within `modules/configuration/config/rules/rules`. Update your rules in Studio. You can use the default rules in the ClaimCenter 6.0.8 folder as a comparison. Compare your custom rules to the new default 6.0.8 versions and update your rules as needed.

You might find it useful to do a bulk comparison of default rules from the base release against the 6.0.8 versions to determine what types of changes Guidewire has made.

To compare rules between versions using the Rule Repository Report

1. If you want to compare default rules only, temporarily remove custom rules from your starting version by moving the `modules/configuration/config/rules` directory to a location outside the ClaimCenter directory.

If you want to compare your custom rules against the ClaimCenter 6.0.8 rules, do not move the `modules/configuration/config/rules` directory from your starting version. However, do remove the `ClaimCenter<base version>` directory from `modules/configuration/config/rules/rules` of the starting version if this directory exists.

2. Open a command window.
3. Navigate to the `bin` directory of your starting version.
4. Enter the following command:

```
gwcc regen-rulereport
```

This command creates a rule repository report XML file in `build/cc/rules`.
5. Append the starting version number to the XML file name.
6. Restore moved directories to the starting version.
7. Temporarily copy `modules/configuration/config/rules` from the target version to a location outside the ClaimCenter directory. Removing this directory prevents your custom rules from being listed in the report. Because the Configuration Upgrade Tool does not modify your custom rules, there is no reason to compare your custom rules in the target configuration.
8. Navigate to the `bin` directory of your target ClaimCenter 6.0.8 version.
9. Enter the following command:

```
gwcc regen-rulereport
```

This command creates a rule repository report XML file in `build/rules`. There is a slight change to the path between the versions.
10. Append the target version number to the XML file name.
11. Restore the `modules/configuration/config/rules` directory to the target version.
12. Open both rule report XML files in a merge tool. You do not merge base rules using the rule repository reports. However, looking at changes that Guidewire has made to the base rules can help you determine the types of changes you must make in your custom rules.

In your merge tool, disable whitespace differences and comments to reduce the amount of inconsequential differences shown between rules.

Update custom rules using Studio. Studio does not compare your rules directly with target rules. However, Studio provides powerful Gosu editing features not available in a standard text editor that can alert you to issues.

When you have finished updating all of your custom rules, delete the ClaimCenter 6.0.8 rules directory from `modules/configuration/config/rules/rules`.

The ClaimCenter 6.0.8 default rules are enabled because some features depend on these rules.

Rules Required for Key Features

Some rules provided with ClaimCenter must be active for certain functionality to work. These rules are listed below by the feature for which they are required. Review these rules and determine which of the applicable functionality you want in your implementation.

Contact Automatic Synchronization

CCL02000 - Suspend AutoSync for Related Contacts

COP01000 - Update Check Address

Catastrophe Bulk Association

CPU09000 - Related to Catastrophe

CPU13000 - Catastrophe History

CLV10000 - Catastrophe

Claim Health Metrics

CPU18000 - Update Claim Health

EPU07000 - Update Claim Health

TPU07000 - Update Claim Health

Deductible Handling

EPU04000 - Update Deductible On Updated Exposure Coverage

EPU05000 - Update Deductible On Updated Coverage Deductible

EPU06000 - Stop Closing Of Exposure With Unpaid Deductible

TPU06000 - Unlink Deductible After Check Denial

Reinsurance

TPU04000 - Reinsurance

ISO Validation

CLV09000 - ISO Validation

Translating New Display Properties and Typecodes

ClaimCenter 6.0.8 adds new display properties and typecodes. If you have defined additional locales, export these new display properties and typecodes to a file, define localized values, and reimport the localized values. If you do not have additional locales defined in your ClaimCenter environment, skip this procedure.

To localize new display properties and typecodes

1. Open Studio from your ClaimCenter 6.0.8 environment by entering the following command from ClaimCenter\bin:

```
gwcc studio
```
2. In the Studio Resources pane, right-click **configuration** → **Typelists**.
3. Select **Export translation file** → **English (US) to locale**, where *locale* is the name of the locale for which you want to translate the new display properties and typecodes. A file browser window opens.
4. Choose a location and name for the exported file.
5. Open the file in a text editor. The first section of the file lists display properties and typecodes that have a localized value. The second section lists display properties and typecodes that do not have a localized value.
6. Specify localized values for the untranslated properties.
7. Save the updated file.
8. In the Studio Resources pane, right-click **configuration** → **Typelists**.

9. Select **Import translation file** → *locale*, where *locale* is the name of the locale for which you want to import the localized display properties and typecodes. A file browser window opens.
 10. Use the file browser window to navigate to the file containing translated display properties and typecodes.
 11. Click **Open**. Studio imports the localized typecodes and display keys. Studio also copies localized display keys to a `modules/configuration/config/locale/locale/display.properties` file.
- After you import the localized typecodes and display keys, you can view them in Studio.

Modifying PCF files, Rules and Libraries for Unused Contact Subtypes

You might not want to use all of the contact subtypes provided with ClaimCenter. However, contact subtypes are referenced within PCF files (particularly in modal pages of the address book tab and ContactCenter), rules, and libraries. Therefore, Guidewire recommends that you do not remove the unused contact subtypes. Instead, modify PCF files, rules and libraries that reference either the specific unused subtypes or the `ContactSubtype` entity itself.

To find contact subtype references

1. Open Guidewire Studio using the `gwcc studio` command in `ClaimCenter/bin`.
2. Right-click **Rule Sets, Classes, or Page Configuration (PCF)** and select **Find in Path**.
3. For **Text to find**, enter `ContactSubtype`.
4. Click **Find**.
5. Repeat this procedure, searching for unused contact subtype names instead of `ContactSubtype`. For example, if you are not using the `Adjudicator` contact subtype, search for `Adjudicator` to see which files, rules and libraries reference this unused subtype.

After you have found all references to the `ContactSubtype` entity and the specific unused contact subtypes, review each case to determine how to proceed.

If the usage is in a range input, as in `AddressBookSearchDV.pcf`, use a filter to exclude the unused contact subtypes from the range input. Alternatively, you can set the filter to only include the contact subtypes that you want to use.

If a menu item uses the contact subtype, as in `AddressBookMenuActions.pcf`, remove the menu item to prevent users from performing actions with the unused contact subtype.

Converting Velocity Templates to Gosu Templates

Guidewire has replaced Velocity templates with Gosu templates. If you have customized templates, you must manually convert these templates to Gosu. This topic outlines steps necessary to upgrade customized templates and maps old functionality to new.

The default templates for Claim and Check duplicate searches have been rewritten. If you are using the default duplicate search templates without modification, you do not need to do anything.

IMPORTANT If you have customized velocity templates, you must manually convert these templates to Gosu. Your customizations to the default templates appear in the Configuration Upgrade Tool under `config/templates/duplicate-search/`. You can compare changes you have made to the default version using a diff tool for reference purposes only.

New Location for Duplicate Search Templates

The duplicate search templates have been renamed and given a new extension. Gosu templates are now first class objects which live in the Gosu classes package structure.

Old Velocity template	New Gosu template	Description
check.vm	gw.duplicaterearch.DuplicateCheckSearchTemplate.gst	Duplicate Check search
claim.vm	gw.duplicaterearch.DuplicateClaimSearchTemplate.gst	Duplicate Claim search

You can access the Gosu duplicate search templates for ClaimCenter 6.0.8 using Guidewire Studio. Open **Configuration** → **Classes** → **gw** → **duplicaterearch**.

Parameters

The `DuplicateCheckSearchTemplate` takes three parameters:

- `DuplicateSearchHelper`, which provides utilities for SQL construction.
- the `Check` to search for duplicates.
- a `checkBeingCloned` parameter. If the `Check` is a clone of an existing check, then this parameter contains the existing `Check`. The search avoids returning the existing `Check` or any of its recurrences as a duplicate. Otherwise, `checkBeingCloned` is `null`.

These parameters differ from the original `check.vm` Velocity template. The original template took a couple of additional parameters: `PayeeTaxIds` and `CheckIdsToIgnore`, a possibly empty list of check ids that are not considered duplicates. The values that used to be in these parameters are now calculated in Gosu within the template, so the logic can be modified if necessary.

The `DuplicateClaimSearchTemplate` takes just two parameters:

- `DuplicateSearchHelper`, which provides utilities for SQL construction.
- the `Claim` to search for duplicates.

This closely mirrors the original parameters to the old `claim.vm` Velocity template, except that the `DuplicateSearchHelper` has changed a little, as described later in this topic.

Key Language Differences

The following table shows some key language differences between Velocity and Gosu.

Area	Velocity	Gosu
Comments	<pre>## This is a comment. *#</pre>	<pre>/* This is a comment that spans multiple lines. */ and // This is a single-line comment</pre>
Initializing a variable	<code>#set(\$myVar = 123)</code>	<code>var myVar = 123</code>
Modifying a variable	<code>#set(\$myVar = "new Value")</code>	<code>myVar = "new Value"</code>

Area	Velocity	Gosu
Conditional expressions	<p>Conditional expressions that evaluate to null or false are equivalent.</p> <p>In the following Velocity example, the if block is only executed if \$myCondition evaluates to true.</p> <pre>#if (\$myCondition) ... #endif</pre> <p>The variable \$myCondition is evaluated to determine whether it is true. This happens under one of two circumstances:</p> <ul style="list-style-type: none"> \$myCondition is a Boolean that has a true value. the value is not null. The condition can be a non-Boolean object that is treated as true if not null and false if null. 	<p>The condition has to be a Boolean or the template does not compile. It is possible for a Boolean to be null, in which case it is treated as false.</p>
If conditional block	<pre>#if (\$myCondition) ... #end</pre>	<pre>if (myCondition) { ... }</pre>
If-else conditional block	<pre>#if (\$myCondition) ... #else ... #end</pre>	<pre>if (myCondition) { ... } else { ... }</pre>

VelocityUtil to DuplicateSearchHelper Transition Table

The support class `com.guidewire.cc.domain.duplicatesearch.VelocityUtil` is removed and superceded by `gw.api.util.DuplicateSearchHelper`. Some methods from `VelocityUtil` have been ported directly over to `DuplicateSearchHelper` and function identically. Other methods have been slightly changed. Some have been removed because they can be replaced by an equivalent Gosu expression or method call. The following table outlines these changes and equivalences. These methods are described in Studio. Use CTRL + Q in Studio to get details.

VelocityUtil Method	Status	Equivalence
<code>public String makeParam(Object value)</code>	unchanged	Identical method and signature on <code>DuplicateSearchHelper</code> .
<code>public String makeParam(String beanPath, Object value)</code>	unchanged	Identical method and signature on <code>DuplicateSearchHelper</code> .
<code>public String makeParam(IEntityPropertyInfo propertyInfo, Object value)</code>	unchanged	Identical method and signature on <code>DuplicateSearchHelper</code> .
<code>public DateTimeUtil getDate()</code>	deleted	Use <code>gw.api.util.DateUtil</code> instead.
<code>public SQLUtil getSql()</code>	deleted	All methods on <code>SQLUtil</code> are now directly available on <code>DuplicateSearchHelper</code> .
<code>public int getArrayLength(Object[] array)</code>	deleted	Use <code>.length</code> property on array object.
<code>public Object getTypeCodeId(String typeListName, String typeCode)</code>	modified	Replaced by <code>DuplicateSearchHelper.getTypeCodeId(TypeKey key)</code> .
<code>IEntityType getEntityIntrinsicType(String name)</code>	deleted	Use Gosu operator <code>typeof</code> .
<code>String escapeSqlString(String str)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).
<code>String getSQLStringValue(Object value)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).
<code>String compareIfEqualTo(Object value)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).

VelocityUtil Method	Status	Equivalence
String <code>compareToIgnoreCase(Object value)</code>	added	Used to be on <code>SQLUtil (getSql())</code> .
String <code>format(String input, Object[] args)</code>	added	Used to be on <code>SQLUtil (getSql())</code> .

One specific example line to change is the following:

Velocity version:

```
VelocityUtil.getEntityIntrinsicType("Person").isAssignableFrom(myClaim.Insured)
```

Gosu version:

```
myClaim.Insured typeis Person
```

The original Velocity version of the line is included in the default duplicate search templates.

Validating the ClaimCenter 6.0.8 Configuration

This topic includes procedures to validate the upgraded configuration.

Using Studio to Validate Files

You can use Studio to validate files and rules without having to start ClaimCenter. Do not start ClaimCenter at this point. Studio can run without connecting to the application server.

To fix all problems in classes and enhancements including libraries, as well as PCF files and typelists:

1. Run the **Validate** option in the Studio **Tools** menu. This identifies all incorrect Gosu syntax in all Studio resources, issuing either a warning or an error.
2. Correct all identified errors with Studio. You can defer fixing warnings.

Starting ClaimCenter and Resolving Errors

IMPORTANT In the process described in this section, do not point the ClaimCenter server at a production database. The goal of this process is to test the configuration upgrade. Create an empty database account and point ClaimCenter to this account for this process. See “Configuring the Database” on page 20 in the *Installation Guide* and “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Upon starting the server for the first time, you might receive errors that prevent the server from starting. In general, fixing errors and starting the server is an iterative process that involves:

1. Start the server for the target configuration.
ClaimCenter encounters a configuration error and shuts down.
2. Copy the error message to a log file.
3. Locate the configuration causing the error - for example, a line of code in a PCF.
4. Comment out the offending line.
After the server starts successfully, look at the log and start solving errors and introducing solutions into the environment. Assign errors to developers on your team.
5. Copy the commented file to the test bed for later analysis.
6. Begin again with step 1. Continue until the server starts successfully.

When the server starts successfully, resolve any remaining issues in the configuration that caused startup errors. Attempt to resolve each error individually and start the server to see if the fix worked.

Building and Deploying ClaimCenter 6.0.8

After you apply and validate an upgrade to the configuration environment, rebuild and redeploy ClaimCenter. Before you begin, make sure you have carefully prepared for this step. In particular, make sure you have updated your infrastructure appropriately.

Review this topic and then rebuild and redeploy ClaimCenter to the application server. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide* of the target version for instructions.

WARNING Do not yet start ClaimCenter. Only package the application file and deploy it to the application server. Starting ClaimCenter begins the database upgrade.

If you have multiple Guidewire products to upgrade, such as ContactCenter, upgrade, build, and deploy each individually before attempting to re-integrate them.

The Build Environment

With the exception of the database configuration, the first time you start the application server use the unmodified `config.xml` and `logging.properties` files provided with the target configuration. After the server starts successfully, you can merge in specific configurations of these files.

If you are using a `build.xml` file to run the `ClaimCenter/bin` build tasks, ensure that it is updated correctly for the target infrastructure. You might encounter problems running build scripts if the data dictionary generation fails due to a PCF validation error. However, this dictionary does not report these errors.

If you encounter build failures due to data dictionary generation, you can comment out this dictionary generation. Then, as you start the server, it reports any PCF configuration errors. After you have corrected PCF configurations, un-comment the dictionary generation and rebuild the application file.

Preserving JAR Files

Place custom JAR files in the `/config/lib` directory. The `build-war`, `build-weblogic-ear`, or `build-websphere-ear` command and subsequent deployment of the packaged application copies the JAR file into the appropriate place for it to be accessed by ClaimCenter. JAR files in this location survive the upgrade process.

Upgrading the ClaimCenter 5.0.x Database

This topic provides instructions for upgrading the ClaimCenter database to ClaimCenter 6.0.8.

Some procedures described in sections within this topic require the ClaimCenter toolkit in the customer configuration. If you do not see the `toolkit/bin` folder in the customer configuration, run `gwcc regen-toolkit` from the `bin` directory. After you have regenerated the toolkit, you can use the system tools on the current customer configuration. You do not need to generate the toolkit for the target configuration. Toolkit utilities in ClaimCenter 6.0.8 are in the `admin/bin` directory.

This topic includes:

- “Upgrading Administration Data for Testing” on page 118
- “Identifying Data Model Issues” on page 120
- “Verifying Batch Process and Work Queue Completion” on page 121
- “Purging Data Prior to Upgrade” on page 121
- “Validating the Database Schema” on page 123
- “Checking Database Consistency” on page 123
- “Creating a Data Distribution Report” on page 123
- “Generating Database Statistics” on page 124
- “Creating a Database Backup” on page 125
- “Updating Database Infrastructure” on page 125
- “Preparing the Database for Upgrade” on page 125
- “Handling Extensions” on page 126
- “Creating Extensions to Preserve Data” on page 127
- “Disabling Encryption for Upgrade Performance” on page 131

- “Setting Linguistic Search Collation for Oracle” on page 132
- “Using the IDatabaseUpgrade Plugin” on page 133
- “Disabling the Scheduler” on page 141
- “Suspending Message Destinations” on page 141
- “Configuring the Database Upgrade” on page 142
- “Starting the Server to Begin Automatic Database Upgrade” on page 145
- “Correcting Issues with Multiple Exposures on Incidents” on page 158
- “Viewing Detailed Database Upgrade Information” on page 160
- “Dropping Unused Columns on Oracle” on page 161
- “Setting Claim Descriptions” on page 161
- “Exporting Administration Data for Testing” on page 162
- “Final Steps After The Database Upgrade is Complete” on page 164

Upgrading Administration Data for Testing

You might want to create an upgraded administration data set for development and testing of rules and libraries with ClaimCenter 6.0.8. You can wait until the full database upgrade is complete and then export the administration data, as described in “Exporting Administration Data for Testing” on page 162. Or, you can upgrade only the administration data to have this data available earlier in the upgrade process. Use the procedure in this section to create an upgraded administration data set before upgrading the full database.

To upgrade administration data

1. Export administration data from your current (pre-upgrade) ClaimCenter production instance:
 - a. Log on to ClaimCenter as a user with the `viewadmin` and `soapadmin` permissions.
 - b. Click the **Administration** tab.
 - c. Choose **Import/Export Data**.
 - d. Select the **Export** tab.
 - e. Select **Admin** from the **Data to Export** dropdown.
 - f. Click **Export**. ClaimCenter exports an `admin.xml` file.
2. On a new pre-upgrade development environment based on your production configuration, create an empty version of `importfiles.txt` in the `modules/configuration/config/import/gen` directory.
3. Create empty versions of the following CSV files:
 - `activity-patterns.csv`
 - `authority-limits.csv`
 - `reportgroups.csv`
 - `roleprivileges.csv`
 - `rolereportprivileges.csv`Leave `roles.csv` as the original complete file.
4. Start the development environment server by opening a command prompt to `ClaimCenter/bin` and entering the following command:

```
gwcc dev-start
```
5. Import this administration data into the development environment.
 - a. Log on to ClaimCenter as a user with the `viewadmin` and `soapadmin` permissions.

- b. Click the **Administration** tab.
 - c. Choose **Import/Export Data**.
 - d. Select the **Import** tab.
 - e. Click **Browse....**
 - f. Select the `admin.xml` file that you exported in step 1.
 - g. Click **Open**.
6. Create a backup of the new development environment database.
7. Create a new database account for the development environment on a database management system supported by ClaimCenter 6.0.8. 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 <http://guidewire.custhelp.com>.
See “Configuring the Database” on page 20 in the *Installation Guide* for instructions to configure the database account.
8. Restore the backup of the database containing the imported administration data into the new database.
9. Connect your upgraded target ClaimCenter 6.0.8 configuration to the restored database.
10. Start the ClaimCenter 6.0.8 server to upgrade the database.
11. Export the upgraded administration data:
 - a. Start the ClaimCenter 6.0.8 server by navigating to `ClaimCenter/bin` and running the following command:
`gwcc dev-start`
 - b. Open a browser to ClaimCenter 6.0.8.
 - c. Log on as a user with the `viewadmin` and `soapadmin` permissions.
 - d. Click the **Administration** tab.
 - e. Choose **Import/Export Data**.
 - f. Select the **Export** tab.
 - g. For **Data to Export**, select **Admin**.
 - h. Click **Export**. Your browser will note that you are opening a file and will prompt you to save or download the file.
 - i. Select to download the `admin.xml` file.
12. Open `admin.xml`.
13. Modify each occurrence of `cc_systemTables` to `systemTables`.
14. Modify the `public-id` of the default organization from `<Organization public-id="default_data:1">` to `<Organization public-id="systemTables:1">`
15. Update all references to the default organization to the new `public-id` by changing each occurrence of `<Organization public-id="default_data:1"/>` to `<Organization public-id="systemTables:1"/>`.
16. Change the root group `public-id` from `<Group public-id="default_data:1">` to `<Group public-id="systemTables:1">`.
17. Change all references to the root group on group users and assignable queues by changing each occurrence of `<Group public-id="default_data:1"/>` to `<Group public-id="systemTables:1"/>`.

18. Change all references to the root group on child groups by changing each occurrence of `<Parent public-id="default_data:1"/>` to `<Parent public-id="systemTables:1"/>`.
19. Save changes to `admin.xml`. You can now import this upgraded administration data into your ClaimCenter 6.0.8 development environments.

Identifying Data Model Issues

Before you upgrade a production database, identify issues with the datamodel by running the database upgrade on an empty database. This process does not identify all possible issues. The database upgrade does not detect issues caused by specific data in your production database. Instead, this procedure identifies issues with the data model.

Complete the following procedure to identify data model issues, and correct any issues on an empty schema. Then, follow the full list of procedures in this topic to upgrade a production database. This list begins with “Verifying Batch Process and Work Queue Completion” on page 121 and finishes with “Final Steps After The Database Upgrade is Complete” on page 164.

To identify data model issues

1. Create an empty schema of your starting version database. You can do this in a development environment by pointing the development ClaimCenter installation at an empty schema and starting the ClaimCenter server. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.
2. Complete the configuration upgrade for data model files in your starting version, according to the instructions in the previous topic. You do not need to complete the merge process for all files. See also “Handling Extensions” on page 126.
3. Create extensions to preserve data if applicable. Review “Creating Extensions to Preserve Data” on page 127.
4. Configure your upgraded development environment to point to the database account containing the empty schema of your old version. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.
5. Start the ClaimCenter server in your upgraded development environment. The server performs the database upgrade to ClaimCenter 6.0.8. See “Starting the Server to Begin Automatic Database Upgrade” on page 145.
6. Check for errors reported during the upgrade process. Resolve any issues before upgrading your production database.

These errors can be issues detected by the database upgrade version checks or issues with the domain and admin graphs. ClaimCenter uses the domain and admin graphs for a number of features including archiving and purging. ClaimCenter 6.0.8 performs a number of graph validation checks when you start the server. The ClaimCenter server does not start if certain graph validation checks fail. In prior versions, graph validation errors did not prevent the server from starting.

See “Graph Validation Checks” on page 64 in the *System Administration Guide* for a description of the graph validation checks that ClaimCenter performs.

See “The Domain Graph” on page 279 in the *Configuration Guide* for a description of the domain and admin graphs. That topic includes information about configuring the data model properly to maintain graphs suitable for ClaimCenter.

You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. For more information, see “Running Custom SQL” on page 134.

If ClaimCenter detects circular references when the server starts, you can use the `IDatabaseUpgrade` plugin to convert foreign keys to edge foreign keys during the major version database upgrade. See “Eliminating Circular References” on page 136.

Verifying Batch Process and Work Queue Completion

All batch processes and distributed work queues must complete before beginning the upgrade. Check the status of batch processes and work queues in your current production environment.

To check the status of batch processes and work queues

1. Log in to ClaimCenter as the superuser.
2. Press **Alt + Shift + T**. ClaimCenter displays the **Server Tools** tab.
3. Click **Batch Process Info**.
4. Select **Any** from the **Processes** drop-down filter.
5. Click **Refresh**.
6. Check the **Status** column for each batch process listed. This list also includes batch processes that are writers for distributed work queues. If any of the batch processes have a **Status** of **Active**, wait for the batch process to complete before continuing with the upgrade.

Purging Data Prior to Upgrade

This topic includes recommendations for purging certain types of data from the database prior to upgrade. Removing unused records can improve the performance of the database upgrade and ClaimCenter.

Purging Old Messages from the Database

Purge completed inactive messages before upgrading the database. Doing so reduces the complexity of the database upgrade. Use the following command from the current (pre-upgrade) customer configuration `toolkit/bin` directory to purge completed messages:

```
messaging_tools -password password -server http://server:port/instance -purge MM/DD/YY
```

Replace `MM/DD/YY` with a date. This tool deletes completed messages that are older than that date. Periodically use this command to purge old messages to prevent the database from growing unnecessarily.

Or, you can use the web service API `IMessageToolsAPI.purgeCompletedMessages(Calendar date)`.

Always perform this message purge on the database before starting ClaimCenter so that the database upgrade does not attempt to convert those rows.

You cannot resend old messages after the upgrade. This is because integrations change and the message payload might be different. It is important that messages that have failed or not yet been consumed finish prior to upgrading.

After you purge completed inactive messages, reorganize the `cc_MessageHistory` table. You might also want to rebuild any indexes on the table. Contact Guidewire Support if you need assistance.

Purging Orphaned Policies from the Database

Each claim has an associated policy record. In ClaimCenter 6.0 and prior versions, if a user refreshed the policy information on a claim, the old policy record remained in the database. If orphaned policies are consuming large amounts of database space, contact Guidewire Support for assistance removing the policies.

Purging Address Correction Records

The `cc_addresscorrection` table stores address corrections returned from geocoding systems. ClaimCenter does not display address corrections by default. However, you might have configured ClaimCenter to expose address corrections to allow users to make corrections.

If you have not configured ClaimCenter to expose address corrections, you can remove the address correction records by truncating the `cc_addresscorrection` table.

If you do have address corrections exposed, you can remove records that have been handled already.

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

ClaimCenter 5.0.6 and higher 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
```

If you are upgrading from a ClaimCenter version prior to 5.0.6, use the following SQL statements in the order shown to remove these items:

```
delete from cc_workflowlog
  where workflow in ( select id from cc_workflow
    where state = ( select id from cctl_workflowstate where typecode="completed" ))
delete from cc_workflowworkitem
  where workflowid in ( select id from cc_workflow
    where state = ( select id from cctl_workflowstate where typecode="completed" ))
delete from cc_workflow
  where state = ( select id from cctl_workflowstate where typecode="completed" ))
```

Try this in a test environment before applying to your production database.

ClaimCenter 6.0 and higher 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 the problem of a large number of workflow log records causing performance issues.

These work queues are described in “Purging Old Workflows and Workflow Logs” on page 57 in the *System Administration Guide*.

Validating the Database Schema

This validation detects the unlikely event that the data model defined by your configuration files has become out of sync with the database schema. While the pre-upgrade server is running, use the `system_tools` command in `admin/bin` of the customer configuration to verify the database schema:

```
system_tools -password password -verifydbschema -server servername:port/instance
```

Correct any validation problems in the database before proceeding. Contact Guidewire Support for assistance.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Checking Database Consistency

ClaimCenter has hundreds of internal database consistency checks. Before upgrading, run consistency checks by executing the following command from the `admin/bin` directory of the customer configuration.

```
system_tools -password password -checkdbconsistency -server servername:port/instance
```

This command takes a long time and could time out. If it does, run the command on subsets of the database instead of the entire database. See the “`system_tools` Command” on page 173 in the *System Administration Guide* for instructions.

You can also trigger database consistency checks to run at server startup by setting `checker=true` in the database block of `config.xml`.

```
<database name="ClaimCenterDatabase" driver="dbcp"
  dbtype="sqlserver" autoupgrade="false" checker="true">
  ...
</database>
```

Each time the server starts with `checker=true`, it will run database consistency checks. These checks might take a long time to run. The server is not available until the consistency checks are completed. Therefore, when you have finished, disable consistency checks for the next server startup by setting `checker=false`.

Run database consistency checks early in the upgrade project. Fix any consistency errors. Continue to periodically run consistency checks and resolve any issues so that your database is ready to upgrade when you begin the upgrade procedure. Consistency issues might take some time to resolve, so begin the process of running consistency checks and fixing issues early.

IMPORTANT Resolve consistency check errors prior to upgrade, especially those surrounding Financials and Incident creation. After resolving any data issues, run the consistency checks again to ensure the database is ready to be upgraded. Contact Guidewire Support for information on how to resolve any consistency issues.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Creating a Data Distribution Report

Generate a data distribution report for the database before an upgrade. Save the output of this report. Run the report again after the upgrade to ensure the distribution is still correct.

Guidewire is very interested in the data distribution of your databases. Guidewire uses these reports to better understand the nature of your database and to optimize ClaimCenter performance. Guidewire appreciate copies of your reports, both before and after upgrades.

You can also use this information to tune the application server cache. See “Understanding Application Server Caching” on page 70 in the *System Administration Guide*.

To create a database distribution report

1. In `config.xml`, set `<param name="EnableInternalDebugTools" value="true"/>`.
2. Start the ClaimCenter application server.
3. Log into ClaimCenter as an administrative user.
4. Type ALT + SHIFT + T while in any screen to reach the **Server Tools** page.
5. Choose **Info Pages** from the **Server Tools** tab.
6. Choose the **Data Distribution** page from the **Info Pages** dropdown.
7. Enter a reason for running the Data Distribution batch job in the **Description** field.
8. On this page, select the **Collect distributions for all tables** radio button and check all checkboxes to collect all distributions.
9. Push the **Submit Data Distribution Batch Job** button on this page to start the data collection.
10. Return to the **Data Distribution** page and push its **Refresh** button to see a list of all available reports. The batch job has completed when the **Available Data Distribution** list on the **Data Distribution** page includes your description.
11. Select the desired report and use the **Download** button to save it zipped to a text file. Unzip the file to view it.

Generating Database Statistics

To optimize the performance of the ClaimCenter database, it is a good idea to update database statistics on a regular basis. Both SQL Server and Oracle can use these statistics to optimize database queries. Before you upgrade, and before you go into production, update the database statistics by running the `maintenance_tools` command from `toolkit/bin` of the customer (pre-upgrade) configuration.

To generate database statistics

1. Run the `maintenance_tools` command to get the proper SQL statements for updating the statistics in ClaimCenter tables:

```
maintenance_tools -getdbstatisticsstatements -password password  
-server http://server:port/instance> db_stats.sql
```
2. Run the resulting SQL statements against the ClaimCenter database.
3. If using an Oracle database, the `maintenance_tools` command creates `db_stats.ora`. Review and then run these statements against the database.

You can configure a SQL Server instance to periodically update statistics using SQL. See your database documentation and “Configuring Database Statistics” on page 53 in the *System Administration Guide* for more information.

The database upgrade can take a long time, and has built-in statistics collection that help you see if any part of the upgrade is slow. Collect these statistics, and compare them to the statistics you collected before the upgrade. The `config.xml` file has parameters that control this statistics collection.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Creating a Database Backup

Prepare the environment so that you can make a total recovery of the original installation if you run into problems during the upgrade.

The first time you start the ClaimCenter server after running the upgrade tool, the server updates the database. During its work, the database upgrader minimizes the logging that it does. For these reasons, back up your database before starting an upgrade. Your pre-upgrade database might not be recoverable after an upgrade.

Updating Database Infrastructure

Before starting the upgrade, update database server software and operating systems as needed to meet the installation requirements of ClaimCenter 6.0.8. 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>.

For SQL Server, after you upgrade the database server software, run the following command to set the compatibility level:

```
ALTER DATABASE databaseName SET COMPATIBILITY_LEVEL = 100
```

Preparing the Database for Upgrade

This topic notes steps to prepare the database for the upgrade process.

Ensuring Adequate Free Space

The database upgrade requires significant free space. Make sure the database has at least 50% of the current database size available as free space.

Disabling Replication

Disable database replication during the database upgrade.

Assigning Default Tablespace (Oracle only)

Set the default tablespace for the database user to the one mapped to the logical tablespace OP in `config.xml`.

The database upgrade creates temporary tables during the upgrade without specifying the tablespace. If the Oracle database user was created without a default tablespace, Oracle by default creates the tables in the SYSTEM tablespace. The Guidewire database user is likely not to have the required quota permission on the SYSTEM tablespace. This results in an error of the type:

```
java.sql.SQLException: ORA-01950: no privileges on tablespace 'SYSTEM'
```

Even if the default tablespace is not SYSTEM, if the Guidewire database user does not have quota permission on the default tablespace, the temporary table creation during upgrade fails.

Using Proper Clock Settings

The `config.xml` file contains several parameters set to AM/PM times (for example, `<param name="BusinessDayEnd" value="5:00 PM"/>`). If your server uses a 24-hour clock, change these parameters to reflect the server clock (`<param name="BusinessDayEnd" value="17:00"/>`).

Handling Extensions

This topic discusses how to handle extensions during the upgrade to ClaimCenter 6.0.8.

Merging Extensions

ClaimCenter 6.0.8 stores extensions in `.eti` and `.etx` files. An `Entity.eti` file defines a new entity. An `Entity.etx` file defines extensions to an existing entity. During the configuration upgrade, an automated step generates `.eti` and `.etx` files from your current `extensions.xml`. The Configuration Upgrade Tool then compares these files against the extension files included with ClaimCenter 6.0.8.

Guidewire often adds indexes to entities in the target configuration to improve the performance of database queries in ClaimCenter 6.0.8. ClaimCenter requires some of these indexes. Guidewire adds required indexes to entity definitions in the data model. Other indexes are recommended for most installations but can be disabled if they negatively impact performance. Guidewire adds optional indexes to entity extensions so you can disable any if necessary.

For example, to improve the performance of the team group activities and activity calendar pages, Guidewire added `Activity.ClaimID` and `Activity.Priority` to the `activityccu4` index on `Activity`. Guidewire also moved this index from the data model to an extension.

Use the Configuration Upgrade Tool to resolve extensions files. When you merge your custom extensions with Guidewire changes, review each new index added by Guidewire. In most cases, include the new index in the merged extension file. You can modify or remove index definitions based on usage in your deployment.

Reviewing Custom Extensions

Generate and review the data dictionary for the target version to identify any custom extensions that are now obsolete due to Guidewire adding a similar field to the base ClaimCenter.

To generate the data dictionary

1. From the command line, navigate to the `bin` directory of the target version.
2. Run the command `gwcc regen-dictionary`.

This command generates the data and security dictionaries in the `build/dictionary` directory of the target version. To view the data dictionary, open `build/dictionary/data/index.html` in a web browser.

Compare the target version data dictionary with the version in your current environment. If you have extensions that are now available as base fields, consider migrating the data in those fields to the base version. Consider whether an extension is still on the appropriate entity. A new entity could be a more appropriate location for the extension. Review key data model changes that might impact your custom extensions.

If you change an extension location or migrate to a new base field, update any PCF, rule or library that references the extension to reference the new location.

Reconciling the Database with Custom Extensions

The extensions defined in .eti and .etx files (previously in extensions.xml) must match the physical database. Delete all physical columns in the database that are not part of the base installation or defined as extensions before starting the server.

Creating Extensions to Preserve Data

The database upgrade removes some tables and columns. If you have data in a table or column that you want to preserve, create an extension as described in the sections within this topic.

The database upgrade looks for specific extensions, and moves data from the table or column to be deleted to the extension before deleting. If there is data in the table or column to be deleted, and no extension defined, an upgrade check reports the issue. The upgrader then stops before it makes any changes to the database.

If you create an extension to preserve data, update references to that data in rules, libraries, PCF files, and so forth to point to the new extension.

Catastrophes

ClaimCenter 6.0 changed from using CatastropheStates to CatastropheZones. CatastropheStates were introduced in ClaimCenter 5.0. By default, a database upgrade trigger converts CatastropheStates to CatastropheZones. See “Converting Catastrophe States to Catastrophe Zones” on page 154. However, you can choose to continue to use CatastropheState instead of the CatastropheZone functionality. See “Retaining Catastrophe States Instead of Catastrophe Zones” on page 127.

If you do decide to use the new functionality, be aware that ClaimCenter 6.0 does not include a Comments column on CatastropheZone. However, ClaimCenter 5.0 included a Comments column on CatastropheState. If you have data in CatastropheState.Comments, either add a Comments column to CatastropheZone to have the upgrader populate the column based on CatastropheState, or remove the data before upgrading. See “Preserving Comments from Catastrophe State” on page 128.

This topic contains two sections that provide procedures for continuing to use CatastropheStates instead of CatastropheZones and for preserving CatastropheState comments if you do take the new functionality.

Retaining Catastrophe States Instead of Catastrophe Zones

You can choose to continue to use CatastropheState instead of the CatastropheZone functionality. To do so, define a CatastropheState extension table, ccx_catastrophestate, as described in the procedure in this section. If the upgrader detects this table in the data model, it renames cc_catastrophestate to ccx_catastrophestate, preserving data in the table. In this case, the upgrade step that converts catastrophe states to catastrophe zones stops after the rename operation and does not perform the conversion.

To preserve the CatastropheState table

1. Create a CatastropheState.eti file in configuration/config/extensions of the target configuration.
2. Define your CatastropheState data model in CatastrophState.eti. For example:

```
<?xml version="1.0"?>
<entity admin="true" desc="Details of states associated to a catastrophe." entity="CatastropheState"
  exportable="true" loadable="true" platform="false" table="catastrophestate" type="joinarray">
  <foreignkey columnName="CatastropheID" exportable="false" fkentity="Catastrophe" name="Catastrophe"
    nullok="false"/>
  <typekey desc="The state associated to the peril" name="State" nullok="true" typelist="State"/>
  <column desc="Comments regarding the state" name="Comments" nullok="true" type="shorttext"/>
  <index desc="Enforce uniqueness of state per catastrophe" name="internal1" unique="true">
    <indexcol keyposition="1" name="CatastropheID"/>
    <indexcol keyposition="2" name="State"/>
  </index>
</entity>
```



```
</index>
</entity>
```

3. Create a `Catastrophe.etx` file in `configuration/config/extensions`.

4. Add the `CatastropheState` array to `Catastrophe` by inserting the following in `Catastrophe.etx`:

```
<!-- Extension to catastrophe table -->
<extension entityName="Catastrophe">
  <array arrayentity="CatastropheState" desc="Details of states associated with a catastrophe."
    exportable="true" name="States" owner="true"/>
</extension>
```

5. Save and close `CatastropheState.eti` and `Catastrophe.etx`.

When you start the server to launch the database upgrade, the upgrader detects the `CatastropheState` extension and renames the existing `cc_CatastropheState` table to `ccx_CatastropheState`. In this case, the upgrader does not convert `CatastropheStates` to `CatastropheZones`.

Update PCF files and rules to reference `CatastropheState` instead of `CatastropheZone`. By default, ClaimCenter 6.0 references `CatastropheZone` or uses a library to get `CatastropheZone` information in the following rules:

- **Validation** → **ClaimValidationRules** → **CLV10000 - Catastrophe** → **CLV10200 Check Loss Date and Zone**
- **Preupdate** → **ClaimPreupdate** → **CPU09000 - Related to Catastrophe**

See “Catastrophes” on page 107 for PCF information.

You must also update the `CatastropheClaimFinder` batch process. This batch process uses the `GW_CatastropheEnhancement` method `findClaimsByCatastropheZone()` to identify claims that have a loss location within the zones of the catastrophe. Use the following procedure to add a `findClaimsByCatastropheState()` method to `GW_CatastropheEnhancement` and update the `CatastropheClaimFinder` batch process to call the new method.

To update the `CatastropheClaimFinder` batch process

1. Open Studio 6.0.8 by entering `gwcc studio` from `ClaimCenter/bin`.
2. In the Studio Resources pane, open `configuration` → `Classes` → `gw` → `entity` → `GW_CatastropheEnhancement`.
3. Add a new method to search for claims based on the catastrophe states. For example:

```
public function findClaimsByCatastropheState() : gw.api.database.IQueryBeanResult<entity.Claim> {
  var q = findClaims()
  if(this.States.Count > 0) {
    q.join("LossLocation").or( \ or1 -> {
      for(state in this.States) {
        or1.compare("State", Equals, state.Code)
      }
    })
  }
  return q.select()
}
```

4. Select **File** → **Save Changes**.
5. In the Studio Resources pane, open `configuration` → `Classes` → `gw` → `util` → `CatastropheClaimFinderBatch`.
6. Within the `doWork()` method, change the line


```
var catClaims = _catastrophe.findClaimsByCatastropheZone()
```

 to


```
var catClaims = _catastrophe.findClaimsByCatastropheState()
```
7. Select **File** → **Save Changes**.

Preserving Comments from Catastrophe State

If you do convert to using `CatastropheZones`, and you have comments on existing `CatastropheStates`, you must create an extension on `CatastropheZone` to preserve the comments.

The ClaimCenter 5.0.x version of CatastropheState included a Comments column. If there is any data in this column, and there is a Comments column on CatastropheZone, the database upgrader copies the data to the CatastropheZone Comments column. The CatastropheZone entity does not have a Comments column by default. If you have comments in CatastropheState, and you want those comments moved to the new CatastropheZone entity, create a Comments column extension on CatastropheZone. Or, if you do not want to preserve the comments from CatastropheState, alter cc_catastrophestate to remove the Comments column before beginning the upgrade. If the upgrader detects data in cc_catastrophestate.Comments, and does not detect a Comments column extension on CatastropheZone, it reports an error message and aborts the upgrade.

To preserve Catastrophe State comments

1. Create a CatastropheZone.etx file in configuration/config/extensions.
2. Add a Comments column to CatastropheZone by inserting the following in CatastropheZone.etx:

```
<!-- Extension to CatastropheZone -->
<extension entityName="CatastropheZone">
  <column desc="Comments regarding the state" name="Comments" nullok="true" type="shorttext"/>
</extension>
```

3. Save and close CatastropheZone.etx.

Deductible Fields

The database upgrade removes deductible-related fields from Claim, Transaction and Payment. See “Removing Unused Deductible Columns” on page 152.

If you want to preserve these fields, add their definitions as extensions to the data model, as described in this topic. If the upgrader finds these fields defined in the data model, it does not remove them or their data during the upgrade.

To preserve deductible fields

1. Copy the Claim.etx file from the cc/config/extensions folder to configuration/config/extensions in the target configuration. If you already have Claim.etx in configuration/config/extensions, skip this step.
2. Open Claim.etx in a text editor.
3. Comment out the sample extensions listed below by adding <!-- before their definitions and --> after, as shown. Leave the index definitions beneath these definitions uncommented.

```
<!--
<column default="foo" desc="varchar(30) extension; default value of 'foo'" name="VarcharExt"
  nullok="true" type="varchar">
  <columnParam name="size" value="30"/>
</column>
<column default="12" desc="integer extension; default value of '12'" name="IntegerExt" nullok="true"
  type="integer"/>
<array arrayentity="GWExtArray" desc="Sample array extension, using an entity type defined only in the
  extensions file." name="GWExtArray"/>
<typekey desc="Sample TypeKey extension." name="TypeKeyExt" nullok="true" typelist="LossType"/>
-->
```

4. Add a definition for a DeductibleStatus typekey, as shown in bold:

```
<!-- Extension to Claim -->
<extension entityName="Claim">
  ...
  <typekey desc="Whether the deductible has been paid." name="DeductibleStatus"
    typelist="DeductibleStatus"/>
  ...
</extension>
```

5. Save and close Claim.etx.
6. Create a Transaction.etx file in configuration/config/extensions.
7. Open Transaction.etx.

8. Add a definition for a DeductiblePaid typekey column.

```
<!-- Extension to Transaction -->
<extension entityName="Transaction">
  <typekey desc="Whether the deductible has been paid." name="DeductiblePaid"
    typelist="YesNo"/>
</extension>
```

9. Save and close Transaction.etx.

10. Create a Payment.etx file in configuration/config/extensions.

11. Open Payment.etx.

12. Add definitions for the following columns:

- DeductibleAmount
- DeductibleSubtracted
- NetAmount
- OriginalAmount

For example:

```
<!-- Extension to Payment -->
<extension entityName="Payment">
  <column desc="Amount subtracted for deductible." name="DeductibleAmount" nullok="true"
    type="money"/>
  <column desc="Has the deductible been paid?" name="DeductibleSubtracted" nullok="true"
    type="bit"/>
  <column desc="Net amount to pay." name="NetAmount" nullok="true"
    type="money"/>
  <column desc="Original payment amount." name="OriginalAmount" nullok="true"
    type="money"/>
</extension>
```

13. Save and close Payment.etx.

Reinsurance Status

The database upgrade removes the ReinsuranceStatus typekey and references to it. See “Removing Reinsurance Status Typekey and References” on page 151.

If you have data in Claim.ReinsuranceStatus that you want to keep, do the following:

- Create a ReinsuranceStatus typekey extension on Claim.
- Add the ReinsuranceStatus typelist back to the ClaimCenter 6.0.8 configuration.

These procedures are described in this topic.

If the upgrader does not detect a ReinsuranceStatus typekey extension, and there is data in Claim.ReinsuranceStatus, a version check reports an error. If there is no data in Claim.ReinsuranceStatus, and the extension is not defined, the upgrader drops the column. If the upgrader does detect the extension, it does not drop the Claim.ReinsuranceStatus column.

To preserve reinsurance status

1. Copy the Claim.etx file from the cc/config/extensions folder to configuration/config/extensions in the target configuration. If you already have Claim.etx in configuration/config/extensions, skip this step.
2. Open Claim.etx in a text editor.
3. If using the copied Claim.etx, comment out the sample extensions listed below by adding <!-- before their definitions and --> after, as shown. Leave the index definitions beneath these definitions uncommented.

```
<!--
<column default="foo" desc="varchar(30) extension; default value of 'foo'" name="VarcharExt"
  nullok="true" type="varchar">
  <columnParam name="size" value="30"/>
</column>
<column default="12" desc="integer extension; default value of '12'" name="IntegerExt" nullok="true"
```

```

    type="integer"/>
    <array entity="GWExtArray" desc="Sample array extension, using an entity type defined only in the
    extensions file." name="GWExtArray"/>
    <typekey desc="Sample TypeKey extension." name="TypeKeyExt" nullok="true" typelist="LossType"/>
-->

```

4. Add a definition for a ReinsuranceStatus typekey, as shown in bold:

```

<!-- Extension to Claim -->
<extension entityName="Claim">
    ...
    <typekey desc="The reinsurance status for a claim." name="ReinsuranceStatus"
    typelist="ReinsuranceStatus"/>
    ...
</extension>

```

5. Save and close Claim.etx.
6. Create a ReinsuranceStatus.tti file in configuration/config/extensions in the target configuration.
7. Open ReinsuranceStatus.tti in a text editor.
8. Add the following text to ReinsuranceStatus.tti.

```

<?xml version="1.0"?>
<typelist
  xmlns="http://guidewire.com/typelists"
  desc=""
  name="ReinsuranceStatus">
  <typecode
    code="Does_Not_Apply"
    desc="Reinsurance does not apply"
    name="Does not apply"
    priority="1"/>
  <typecode
    code="Evaluating"
    desc="Evaluating reinsurance potential"
    name="Evaluating"
    priority="2"/>
  <typecode
    code="Pursuing"
    desc="Pursuing reinsurance"
    name="Pursuing"
    priority="3"/>
</typelist>

```

9. Save and close ReinsuranceStatus.tti.

Disabling Encryption for Upgrade Performance

ClaimCenter 6.0.8 defines a column override for Contact.TaxID. This column override enables encryption on Contact.TaxID. The database upgrade encrypts the TaxID values one by one. This process slows the database upgrade, particularly for implementations with a large number of contacts defined. You can disable the encryption to improve performance of the database upgrade. Then, in a subsequent maintenance window, you can enable encryption for the TaxID field.

To disable encryption of the contact tax ID

1. Open Studio 6.0.8 by entering the following command from the ClaimCenter 6.0.8 ClaimCenter\bin folder:
gwcc studio
2. In the Studio Resources pane, open Data Model Extensions → extensions.
3. Double-click Contact.etx.
4. Delete the following text from Contact.etx:

```

<column-override
  name="TaxID">
  <columnParam
    name="encryption"
    value="true"/>
</column-override>

```

5. Click **Yes** when Studio asks if you would like to edit `Contact.etx`. Studio moves the file to the configuration module.
6. Click **File** → **Save Changes**.

After the database upgrade, you can add the column override back to `Contact.etx`. The next time you start the server, ClaimCenter encrypts each `Contact.TaxID`.

Setting Linguistic Search Collation for Oracle

IMPORTANT This section applies for Oracle implementations only. On SQL Server, ClaimCenter uses the collation setting of the database server. Database searching on SQL Server is always case-insensitive, and obeys the rules of the Windows collation set on the database. This section does not apply for SQL Server implementations.

WARNING Oracle Java Virtual Machine (JVM) must be installed on all Oracle databases hosting ClaimCenter. The only exception is when the ClaimCenter application locale is English and you only require case-insensitive searches. Ensure that Oracle initialization parameter `java_pool_size` is set to a value of above 50 MB.

You can specify how you want ClaimCenter to collate search results. The `strength` attribute of the `LinguisticSearchCollation` element of `GWLocale` for the default locale in `localization.xml` specifies how ClaimCenter sorts search results. You can set the `strength` to `primary` or `secondary`.

With `LinguisticSearchCollation strength` set to `primary`, ClaimCenter searches results in a case-insensitive and accent-insensitive manner. ClaimCenter considers an accented character equal to the unaccented version of the character if the `LinguisticSearchStrength` for the default application locale is set to `primary`. For example, with `LinguisticSearchCollation strength` set to `primary`, ClaimCenter treats “Reneé”, “Renee”, “renee” and “reneé” the same.

With `LinguisticSearchCollation strength` set to `secondary`, ClaimCenter searches results in a case-insensitive, accent-sensitive manner. ClaimCenter does not consider an accented character equal to the unaccented version of the character if the `LinguisticSearchCollation strength` for the default application locale is set to `secondary`. For example, with `LinguisticSearchCollation strength` set to `secondary`, a ClaimCenter search treats “Renee” and “renee” the same but treats “Reneé” and “reneé” differently. By default, ClaimCenter uses a `LinguisticSearchCollation strength` of `secondary`, for case-insensitive, accent-sensitive searching.

For Oracle, the following rules apply:

- **Case:** All searches ignore the case of the letters, whether `LinguisticSearchCollation strength` is set to `primary` or `secondary`. “McGrath” equals “mcgrath”.
- **Punctuation:** Punctuation is always respected, and never ignored. “O'Reilly” does not equal “OReilly”.
- **Spaces:** Spaces are respected. “Hui Ping” does not equal “HuiPing”.
- **Accents:** An accented character is considered equal to the unaccented version of the character if `LinguisticSearchCollation strength` is set to `primary`. An accented character is not equal to the unaccented version if `LinguisticSearchCollation strength` is set to `secondary`.

Japanese only

- **Half Width/Full Width:** Searches under a Japanese locale always ignore this difference.
- **Small/Large Kana:** Japanese small/large letter differences are ignored only when `LinguisticSearchCollation strength` is set to `primary`, meaning accent-insensitive.
- **Katakana/Hiragana sensitivity:** Searches under a Japanese locale always ignore this difference.
- The long dash character is always ignored.

- Soundmarks (`` and °) are only ignored if `LinguisticSearchCollation` strength is set to primary.

German only

- Vowels with an umlaut compare equally to the same vowel followed by the letter e. Explicitly, “ä”, “ö”, “ü” are treated as equal to “ae”, “oe” and “ue”.
- The Eszett, or sharp-s, character "ß" is treated as equal to “ss”.

ClaimCenter populates denormalized values of searchable columns to support the search collation. For example, with `LinguisticSearchCollation` strength set to primary, ClaimCenter stores the value “Reneé”, “Renee”, “renee” and “reneeé” in a denormalized column as “renee”. With `LinguisticSearchCollation` strength set to secondary, ClaimCenter stores a denormalized value of “renee” for “Renee” or “renee” and stores “reneeé” for “Reneeé” or “reneeé”. Japanese and German locales make additional changes when storing values in denormalized columns in order to conform to the rules listed previously for those locales.

Any time you change the `LinguisticSearchCollation` strength and restart the server, ClaimCenter repopulates the denormalized columns. Previous versions of ClaimCenter populated the denormalized columns with lowercase values for case-insensitive search, equivalent to setting `LinguisticSearchCollation` strength to secondary. If you set `LinguisticSearchCollation` strength to primary, ClaimCenter repopulates the denormalized columns, substituting any accented characters for their base equivalents. This process can take a long time, depending on the amount of data. Therefore, if you want to change `LinguisticSearchCollation` strength to primary, you might want to do so after the database upgrade. If you are concerned about the duration of the database upgrade, you can change your search collation settings after the upgrade. During a maintenance period, change `LinguisticSearchCollation` strength to primary and restart the server to repopulate the denormalized columns.

For Japanese locales, the ClaimCenter database upgrade from a prior major version repopulates the denormalized columns regardless of the `LinguisticSearchCollation` strength value. ClaimCenter must repopulate the denormalized columns for Japanese locales to have search results obey the Japanese-only rules listed previously.

Using the IDatabaseUpgrade Plugin

The `IDatabaseUpgrade` plugin interface provides hooks for custom code that you want to run during the database upgrade. You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. You can also use the `IDatabaseUpgrade` plugin to eliminate circular references in your data model by converting specific foreign keys to edge foreign keys.

WARNING The `IDatabaseUpgrade` plugin runs any time that the server determines that a database upgrade is necessary, whether for a major or minor version. Therefore, use conditionals within your `preUpgrade` and `postUpgrade` methods and `get EdgeForeignKeyUpgrades` property to control execution of your custom code. Test these conditionals thoroughly to ensure that the custom code you define within the methods runs appropriately. Remove or disable the `IDatabaseUpgrade` plugin once you have completed the upgrade.

The `IDatabaseUpgrade` plugin interface includes the following properties:

```
get BeforeUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
get AfterUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
```

These properties are for making custom upgrades to archived XML entities. This is only applicable to implementations that have enabled archiving. If you do not have archiving enabled, return an empty list for these properties.

```
override property get BeforeUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {
```

```

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }

    override property get AfterUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }

```

If you make custom data model changes within ClaimCenter and enable archiving, you can use these properties to define changes for the upgrader to make to the archived XML. See “Upgrading Archived Entities” on page 139.

Running Custom SQL

You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. For example, you might fix a consistency check failure issue, correct issues reported by version checks, or delete a custom extension that you are no longer using.

The `IDatabaseUpgrade` plugin interface contains method signatures for two methods that you must define in your plugin. These signatures are:

- `function preUpgrade(context : IUpgradeContext)`
- `function postUpgrade(context : IUpgradeContext)`

The `preUpgrade` method runs before version checks, so you can include any SQL to resolve version check issues in the `preUpgrade` method. You can also write SQL to resolve consistency check issues among other operations.

The `postUpgrade` method runs after the database upgrade version triggers. You can incorporate SQL in the `postUpgrade` method to move data into tables or columns that did not exist prior to upgrading.

The `IDatabaseUpgrade` interface also contains an `Iterable` property used for eliminating circular references. You must define this property. If you are not using this feature of the `IDatabaseUpgrade` plugin, you can define an empty `Iterable` as follows:

```

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
        return {}
    }

```

Any custom SQL that you write must be idempotent. That is, the SQL must be designed so that it can run against the database multiple times without corrupting data. The `IDatabaseUpgrade` plugin runs each time the database is upgraded, whether for a major or minor version of ClaimCenter or for a data model change such as a new extension. Check the version of the database in your `preUpgrade`, `postUpgrade` methods, and `get EdgeForeignKeyUpgrades` property to ensure that your custom SQL is executed appropriately. Guidewire recommends that you create a custom method to check the database version.

For ClaimCenter 5.0 you can determine the database major and minor version by opening `modules\cc\config\metadata\metadataproperties.xml` and checking the value of the `majorVersion` and `minorVersion` properties.

For 6.0 versions of ClaimCenter, the internal major and minor version numbers are listed in `modules\cc\config\metadata\metadata.properties`.

You can also determine the major and minor version of a ClaimCenter instance by reading the messages reported by the server during startup.

```

[java] serverName timestamp INFO Validating main database connections have all required properties
[java] serverName timestamp INFO Checking if existing database version is current...
[java] serverName timestamp INFO Datamodel version (major, minor, platform) is up to date.

```

This message reports the internal major, minor and platform version numbers. Only the major and minor version are of interest for writing custom SQL. Note that these are internal version numbers and do not match the commonly used external version number, such as 6.0.8.

To run custom SQL

1. Create a new Gosu class that implements `IDatabaseUpgrade`. If you already created a Gosu class to eliminate circular references, you can use the `preUpgrade` and `postUpgrade` methods of that class. The class you create must define the `preUpgrade` and `postUpgrade` methods and the `get EdgeForeignKeyUpgrades` property.

The following example demonstrates using the `preUpgrade` method to set `Claim.DeductibleStatus`, `Claim.ReinsuranceStatus` and `Claim.LossLocationID` to null:

```
package gw.myplugins.upgrade.impl

uses gw.plugin.upgrade.IDatabaseUpgrade
uses gw.plugin.upgrade.IUpgradeContext
uses java.lang.Iterable
uses gw.plugin.upgrade.EdgeForeignKeyUpgradeInfo
uses gw.api.archiving.upgrade.ArchivedDocumentUpgradeVersionTrigger

class DatabaseUpgradeImpl implements IDatabaseUpgrade {

    construct() {
    }

    // custom function to determine if DB version is prior to ClaimCenter 6.0 (DB version 8).
    // MajorVersion of CC 5.x is 7, MajorVersion for CC 6.x is 8

    private function isMajorVersionLessThan8(upgradeContext : IUpgradeContext) : boolean {

        //if null upgrade context is sent, return false and log reason
        if (upgradeContext == null) {
            //printing instead of logging because logs were not shown in console
            print("databaseupgrade.isMajorVersionLessThan8() - null upgradeContext sent")
            return false
        }

        //try to access major version, was throwing exception
        try {

            if (upgradeContext.CurrentMajorVersion < 8) {
                return true
            }

        } catch(e) {

            //swallow exception and return false
            print("databaseupgrade.isMajorVersionLessThan8() - accessing " +
                "upgradeContext.CurrentMajorVersion threw exception. Swallowing following exception" +
                "and returning false.")

            e.printStackTrace()

            return false
        }

        //All other conditions, return false
        return false
    }

    override function preUpgrade(upgradeContext : IUpgradeContext) {

        // first check the major version of the database to determine if scripts need to be run.
        if (isMajorVersionLessThan8(upgradeContext)) {

            // DB will be upgraded to CC6.0, execute pre-upgrade scripts
            upgradeContext.update("Set DeductibleStatus and ReinsuranceStatus to null",
                "UPDATE CC_CLAIM " +
                "SET DeductibleStatus = null, ReinsuranceStatus = null " +
                "WHERE DeductibleStatus is not null OR ReinsuranceStatus is not null")

            upgradeContext.update("Null out LossLocationID",
                "UPDATE CC_CLAIM " +
                "SET LossLocationID = null " +
                "WHERE LossLocationID is not null")

        }
    }
}
```



```

    }

    override function postUpgrade(upgradeContext : IUpgradeContext) {

        // first check the major version of the database to determine if scripts need to be run.
        if (isMajorVersionLessThan8(upgradeContext)) {

            ///## todo: Implement

        }
    }

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
        // If not applicable, return empty iterable.
        return {}
    }

    override property get BeforeUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }

    override property get AfterUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }
}

```

2. Implement the `IDatabaseUpgrade` plugin with the new class. If you already implemented this class to eliminate circular references, skip this step.
 - a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
 - b. In the Studio **Resources** pane, expand **Plugins** → **gw** → **plugin** → **upgrade**.
 - c. Right-click `IDatabaseUpgrade` and select **Implement**.
 - d. On the `IDatabaseUpgrade` tab, click **Add....**
 - e. Select **Gosu**.
 - f. Enter your class, including the package.
 - g. Select **File** → **Save Changes**.

When you start the server to perform the database upgrade from a prior major version, the upgrade calls the plugin and runs your custom `preUpgrade` and `postUpgrade` methods.

Eliminating Circular References

ClaimCenter 6.0.5 and newer enforces data model integrity more strictly than previous versions. ClaimCenter runs a series of checks on the data model during server startup. If any of these checks fails, ClaimCenter reports an error and does not start. One of these checks looks for circular references in the data model.

A circular reference exists when an entity points to an entity of the same type through one or more foreign keys. The circular reference can be simple, in which the entity has a foreign key directly to the same entity. Or, the reference can be more complex, in which an entity has a foreign key to another entity that directly or indirectly references the original entity.

To resolve circular references in the data model, use the `IDatabaseUpgrade` plugin to convert foreign keys causing circular references to edge foreign keys. This plugin enables you to specify the entity, existing foreign

key and the edge foreign key to create. The plugin is called during the database upgrade from a prior major version.

The interface also contains an `Iterable` property used for eliminating circular references. You must define this property. If you are not using this feature of the `IDatabaseUpgrade` plugin, you can define an empty `Iterable` as follows:

```
override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
    return {
    }
}
```

The `IDatabaseUpgrade` plugin interface also contains method signatures for methods that you must define in your plugin. These signatures are:

- function `preUpgrade(context : IUpgradeContext)`
- function `postUpgrade(context : IUpgradeContext)`

If you do not want to run custom SQL before or after the upgrade, you can create these methods with no operations defined within them.

To convert foreign keys to edge foreign keys

1. Change the data model to replace the foreign key with an edge foreign key.
 - a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
 - b. In the Studio **Resources** pane, expand **Data Model Extensions** → **extensions**.
 - c. Double-click *EntityName.etx* to open the file defining the entity extension.
 - d. Change the `foreignKey` element to `edgeForeignKey`.
 - e. Add an `edgeTableName` attribute and set the value to a name for the edge table, such as `edgeTableName="entitynameedge"`.
 - f. Remove the `columnName` attribute.
 - g. If you want the contents of the edge table to be part of the domain graph, add the following subelement to `edgeForeignKey`:


```
<implementsEntity name="Extractable"/>
```

 Be sure to move the closing tag of the `edgeForeignKey` element if necessary to encapsulate the subelement.
 - h. Select **File** → **Save Changes**.
 - i. Repeat steps c through h for each foreign key that you want to convert to an edge foreign key.
2. Create a new Gosu class that implements `IDatabaseUpgrade`. If you already created a Gosu class to run custom SQL before or after the upgrade, you can use the `get EdgeForeignKeyUpgrades` property of that class. The class you create must define the `get EdgeForeignKeyUpgrades` property and the `preUpgrade` and `postUpgrade` methods. If you do not want to run custom SQL before or after the upgrade, you can create these methods with no operations defined within them. For example:

```
package gw.myplugins.upgrade.impl
uses gw.plugin.upgrade.IDatabaseUpgrade
uses gw.plugin.upgrade.IUpgradeContext
uses java.lang.Iterable
uses gw.plugin.upgrade.EdgeForeignKeyUpgradeInfo
uses gw.api.archiving.upgrade.ArchivedDocumentUpgradeVersionTrigger

class DatabaseUpgradeImpl implements IDatabaseUpgrade {

    construct() {
    }

    private var _upgradeContext : IUpgradeContext

    // custom function to determine if DB version is prior to ClaimCenter 6.0 (DB version 8).
    // MajorVersion of CC 5.x is 7, MajorVersion for CC 6.x is 8
```

```

private function isMajorVersionLessThan8(upgradeContext : IUpgradeContext) : boolean {
    _upgradeContext = upgradeContext

    //if null upgrade context is sent, return false and log reason
    if (upgradeContext == null) {
        //printing instead of logging because logs were not shown in console
        print("databaseupgrade.isMajorVersionLessThan8() - null upgradeContext sent")
        return false
    }

    //try to access major version, was throwing exception
    try {
        if (upgradeContext.CurrentMajorVersion < 8) {
            return true
        }
    } catch(e) {
        //swallow exception and return false
        print("databaseupgrade.isMajorVersionLessThan8() - accessing" +
            "upgradeContext.CurrentMajorVersion threw exception. Swallowing following exception" +
            "and returning false.")

        e.printStackTrace()

        return false
    }

    //All other conditions, return false
    return false
}

override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
    // Check the major version of the database to determine which edge foreign keys must be converted.
    // MajorVersion of CC5.x is 7, MajorVersion for CC6.x is 8
    if (isMajorVersionLessThan8(_upgradeContext)) {
        return {
            // add these as necessary
            // new EdgeForeignKeyUpgradeInfo("foreignKeyColumn", entityName, "edgeForeignKey")
        }
    }
    else {
        return {}
    }
}

override function preUpgrade(p0 : IUpgradeContext) {
}

override function postUpgrade(p0 : IUpgradeContext) {
}

override property get BeforeUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {
    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();
    return list;
}

override property get AfterUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {
    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();
    return list;
}
}

```

Add a new `EdgeForeignKeyUpgradeInfo("foreignKeyColumn", entityName, "edgeForeignKey")` line to the `get EdgeForeignKeyUpgrades()` definition for each foreign key that you want to convert. Separate these lines with commas.

3. Implement the `IDatabaseUpgrade` plugin with the new class. If you already implemented this class to run custom SQL before or after the database upgrade, skip this step.

- a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
- b. In the Studio **Resources** pane, expand **Plugins** → **gw** → **plugin** → **upgrade**.
- c. Right-click `IDatabaseUpgrade` and select **Implement**.
- d. On the `IDatabaseUpgrade` tab, click **Add...**
- e. Select **Gosu**.
- f. Enter your class, including the package.
- g. Select **File** → **Save Changes**.

When you start the server to perform the database upgrade from a prior major version, the upgrade calls the plugin and converts the foreign keys to edge foreign keys.

Upgrading Archived Entities

ClaimCenter 6.0.5 and newer include a file-based archiving feature. If you implement archiving, and you make custom data model changes, then you can upgrade the archived XML using the `IDatabaseUpgrade` plugin.

Simple data model changes do not require a custom trigger. These include:

- Adding a new entity
- Updating denormalization columns
- Adding editable columns such as `updatetime`
- Adding new columns
- Changing the nullability of a column

More complex transformations or those that could result in loss of data require a version trigger. These include:

- changing a datatype (other than just length)
- migrating data from one table or column to another
- dropping a column
- dropping a table
- renaming a column
- renaming a table

ClaimCenter upgrades an archived entity as the entity is restored.

The `IDatabaseUpgrade` plugin interface includes the following properties:

```
get BeforeUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
get AfterUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
```

You can define custom `ArchivedDocumentUpgradeVersionTrigger` entities to modify archived XML. The `ArchivedDocumentUpgradeVersionTrigger` is an abstract class that you can extend to create your custom triggers.

Define the constructor of your custom `ArchivedDocumentUpgradeVersionTrigger` to call the constructor of the superclass and pass it a numeric value. For example:

```
construct() {
    super(171)
}
```

This numeric value is the extension version to which your trigger applies. If you run the upgrade against a database with a lower extension version, then your custom trigger is called. The current extension version is defined in `modules\cc\config\extensions\extensions.properties`.

Provide an override definition of the `get Description` property to return a `String` that describes the actions of your trigger.

Provide an override definition for the `execute` function to define the actions that you want your custom trigger to make on archived XML.

When the upgrade executes your custom trigger, it wraps each XML entity in an `IArchivedEntity` object. Each typekey is wrapped in an `IArchivedTypekey` object. The upgrade operates on a single XML document at a time.

`ArchivedDocumentUpgradeVersionTrigger` provides the following key operations:

- `getArchivedEntitySet(entityName : String)` – returns an `IArchivedEntitySet` object that contains all `IArchivedEntity` objects of the given type in the XML document.

`IArchivedEntitySet` provides the following key methods:

- `rename(newEntityName : String)` – renames all rows in the set to the new name.
- `delete()` – deletes all rows in the set.
- `search(predicate : Predicate<IArchivedEntity>)` – returns a `List` of `IArchivedEntity` objects that match the given predicate.
- `create(referenceInfo : String, properties : List<Pair<String, Object>>)` – returns a new `IArchivedEntity` with the given properties.

`IArchivedEntity` provides the following key methods:

- `delete()` – deletes just this row.
- `getPropertyValue(propertyName : String)` – returns the value of the property of the given name. If the property value is a reference to another entity, this method returns an `IArchivedEntity`.
- `move(newEntityName : String)` – moves this to a new entity type of the given name. The type is created if it did not exist. You must add any required properties to the type.
- `updatePropertyValue(propertyName : String, newValue : String)` – updates the property of the given name to the given value.
- `getArchivedTypekeySet(typekeyName : String)` – returns an `IArchivedTypekeySet` object that contains all `IArchivedTypekey` objects in the given typelist.
- `getArchivedTypekey(typelistName : String, code : String)` – Returns an `IArchivedTypekey` representing the typekey.

`IArchivedTypekey` provides the following key method:

- `delete()` – deletes the typekey from the XML.

More methods are available for `IArchivedEntitySet`, `IArchivedEntity` and `IArchivedTypekey`. See the Gosu documentation for a full listing. Generate the Gosu documentation by navigating to the ClaimCenter bin directory and entering the following command:

```
gwcc regen-gosudoc
```

Incremental Upgrade

When ClaimCenter archives an entity, it records the current data model version on the entity. ClaimCenter upgrades an archived entity as the entity is restored. The upgrader executes the necessary archive upgrade triggers incrementally on each archived XML entity, according to the data model version of the archived entity.

An entity archived at one version might not be restored and upgraded until several intermediate data model upgrades have been performed. Therefore, do not delete your custom upgrade triggers.

Consider the following situation. Note that this example is for demonstration purposes only. The version numbers included do not represent actual ClaimCenter versions but are included to explain the incremental upgrade process. Each '+' after a version number indicates a custom data model change.

Guidewire data model changes

v6.0.0 – Entity does not have column X.

v6.0.1 – Guidewire adds column X to the entity with a default value of 100.

v6.0.2 – Guidewire updates the default value of column X to 200.

Implementation #1 upgrade path

v6.0.0+ – Start with version 6.0.0 data model with custom changes.

v6.0.0++ – More custom data model changes.

v6.0.0+++ – More custom data model changes.

v6.0.2+ – column X introduced with a default value of 200.

Result of upgrade of a claim archived at v6.0.0+: column X has value 200.

In this situation, version 6.0.1 was skipped in the upgrade path. Therefore, column X is added with the default value of 200 that it has in version 6.0.2.

Implementation #2 upgrade path

v6.0.0+ – Start with version 6.0.0 data model with custom changes.

v6.0.0++ – More custom data model changes.

v6.0.1+ – column X introduced with a default value of 100.

v6.0.2+ – column X already exists.

Result of upgrade of a claim archived at v6.0.0+: column X has value 100.

In this situation, column X is added in version 6.0.1 with the default value of 100. During the upgrade from version 6.0.1 to version 6.0.2, column X already exists. Therefore, the upgrader does not add the column. A custom trigger would be required to update the value of X from 100 to 200 during the upgrade from version 6.0.1 to 6.0.2.

Disabling the Scheduler

Before you start the server to upgrade the database, disable the scheduler for batch processes and work queues. Disabling the scheduler prevents batch processes and work queues from launching immediately after the database upgrade.

To disable the scheduler

1. Open the ClaimCenter 6.0.8 `config.xml` file in a text editor.

2. Set the `SchedulerEnabled` parameter to `false`.

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

3. Save `config.xml`.

After you have successfully upgraded the database, you can enable the scheduler by setting `SchedulerEnabled` to `true`. This can be accomplished by performing the database upgrade using a WAR or EAR file that has the `SchedulerEnabled` parameter to `false`. After the upgrade is complete and verified, stop the server and deploy a new WAR or EAR file that differs from the first only by having `SchedulerEnabled` set to `true`. Finally, restart the server to activate the scheduler.

Suspending Message Destinations

Suspend all event message destinations before you upgrade the database to prevent ClaimCenter from sending messages until you have verified a successful database upgrade.

To suspend message destinations

1. Start the ClaimCenter server for the pre-upgrade version.
2. Log in to ClaimCenter with an account that has administrative privileges, such as the superuser account.
3. Click the **Administration** tab.
4. Click **Event Messages**.
5. Select the check box to the left of the **Destination** column to select all message destinations.
6. Click **Suspend**.

Resume messaging after you have verified a successful database upgrade.

Configuring the Database Upgrade

You can set parameters for the database upgrade in the ClaimCenter 6.0.8 `config.xml` file. The `<database>` block in `config.xml` contains parameters for database configuration, such as connection information. The `<database>` block contains an `<upgrade>` block that contains configuration information for the overall database upgrade. The `<upgrade>` block also contains a `<versiontriggers>` element for configuring general version trigger behavior and can contain `<versiontrigger>` elements to configure each version trigger.

This topic describes the parameters you can set for the database upgrade. For general database connection parameters, see “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Configuring Column Removal on Oracle

The database upgrade removes some columns. For Oracle, you can configure whether the removed columns are dropped immediately or are marked as unused. Marking a column as unused is a faster operation than dropping the column immediately. However, because these columns are not physically dropped from the database, the space used by these columns is not released immediately to the table and index segments. You can drop the unused columns after the upgrade during off-peak hours to free the space. Or, you can configure the database upgrade to drop the columns immediately during the upgrade. By default, the ClaimCenter database upgrade marks columns as unused.

To configure the ClaimCenter upgrade to drop columns immediately during the upgrade, set the `oracleMarkColumnsUnused` attribute of the `<upgrade>` block to `false`. For example:

```
<database ...>
...
  <upgrade oracleMarkColumnsUnused="false">
    ...
  </upgrade>
</database>
```

Configuring Index Creation Parallelism on Oracle

You can configure the database upgrade to create indexes in parallel on Oracle databases. This can potentially reduce the time to upgrade the database if there are sufficient resources available on the database server. By default, indexes are not created in parallel on Oracle. Enable parallel index creation by setting the `createIndexInParallel` attribute of the `<upgrade>` block in the `<database>` block of `config.xml`. For example:

```
<database ...>
...
  <upgrade createIndexInParallel="2">
    ...
  </upgrade>
</database>
```

Set a numeric value for `createIndexInParallel` to specify the degree of parallelism for index creation during the upgrade. Or, set `value="default"` to have Oracle automatic parallel tuning determine the degree to use.

Oracle determines the degree based on the number of CPUs and the value set for the Oracle parameter `PARALLEL_THREADS_PER_CPU`.

Enabling Collection of Tablespace Usage and Object Size

To enable collection of tablespace usage and object size data on Oracle, set the `collectstorageinstrumentation` attribute of the `<upgrade>` block to `true`. For example:

```
<database ...>
...
<upgrade collectstorageinstrumentation="true">
...
</upgrade>
</database>
```

A value of `true` enables ClaimCenter to collect tablespace usage and size of segments such as tables, indexes and LOBs (large object binaries) before and after the upgrade. The values can then be compared to find the utilization change caused by the upgrade.

Enabling Oracle Logging

You can enable logging of direct insert and create index operations during the database upgrade by setting `allowUnloggedOperations` to `false` in the `<upgrade>` block. For example:

```
<database ...>
...
<upgrade allowUnloggedOperations="false">
...
</upgrade>
</database>
```

The default is to run these statements with the `NOLOGGING` option. Although Guidewire recommends that you backup the database before and after the upgrade, there could be reasons to log all operations. Some examples include Reporting, Disaster Recovery through Standby databases and Oracle Dataguard. To enable logging of direct insert and create index operations, set `allowUnloggedOperations` to `false`.

Storing Temporary Sort Results in tempdb

For SQL Server databases, you can specify to store temporary sort results in `tempdb` by setting the `sqlserverCreateIndexSortInTempDB` attribute of the `upgrade` block to `true`. By using `tempdb` for sort runs, disk input and output is typically faster, and the created indexes tend to be more contiguous. By default, `sqlserverCreateIndexSortInTempDB` is `false` and sort runs are stored in the destination filegroup.

If you set `sqlserverCreateIndexSortInTempDB` to `true`, you must have enough disk space available to `tempdb` for the sort runs, which for the clustered index include the data pages. You must also have sufficient free space in the destination filegroup to store the final index structure, because the new index is created before the old index is deleted. Refer to <http://msdn.microsoft.com/en-us/library/ms188281.aspx> for details on the requirements to use `tempdb` for sort results.

Adjusting Commit Size for Encryption

You can adjust the commit size for rows requiring encryption by setting the `encryptioncommitsize` attribute to an integer in the `<upgrade>` block. For example:

```
<database ...>
...
<upgrade encryptioncommitsize="5000">
...
</upgrade>
</database>
```

If ClaimCenter encryption is applied on one or more attributes, the ClaimCenter database upgrade commits batches of encrypted values. The upgrade commits `encryptioncommitsize` rows at a time in each batch. The

default value of `encryptioncommitsize` varies based on the database type. For Oracle, the default is 10000. For SQL Server, the default is 100.

Specifying Filegroup to Store Sort Results for Clustered Indexes

For SQL Server databases, a version trigger recreates non-clustered backing indexes for primary keys as clustered indexes. This change improves the performance of claim archiving and claim purging operations.

Before recreating the indexes, the version trigger automatically drops (and later rebuilds) any referencing foreign keys and drops any clustered indexes on tables with a primary key.

If you are using filegroups, the upgrade recreates the clustered index in the OP filegroup. By default, the upgrade also stores the intermediate sort results that are used to build the index in the OP filegroup. You can configure the upgrade to instead use the tempdb filegroup for the intermediate sort results.

If you want the upgrade to store the intermediate sort results in the tempdb filegroup, set the `sqlserverCreateIndexSortInTempDB` attribute of the upgrade element to `true`.

```
<database ...>
...
  <upgrade sqlserverCreateIndexSortInTempDB="true" />
  ...
</upgrade>
</database>
```

This option increases the amount of temporary disk space that is used to create an index. However, it might reduce the time that is required to create or rebuild an index when tempdb is on a different set of disks from that of the user database.

By default, `sqlserverCreateIndexSortInTempDB` is `false`.

Configuring Version Trigger Elements

The database upgrade executes a series of version triggers that make changes to the database to upgrade between versions. You can set some configuration options for version triggers in `config.xml`. Normally, the default settings are sufficient. Change these settings only while investigating a slow database upgrade.

The `<database>` element in `config.xml` contains an `<upgrade>` element to organize parameters related to database upgrades. Included in the `<upgrade>` element is a `<versiontriggers>` element, as shown below:

```
<database ...>
  <param ... />
  <upgrade>
    <versiontriggers dbmsperfinfothreshold="600" degreeofparallelismforinsertselects="2"/>
  </upgrade>
</database>
```

The `<versiontriggers>` element configures the instrumentation of version triggers. This element has two attributes: `dbmsperfinfothreshold` and `degreeofparallelismforinsertselects`.

The `dbmsperfinfothreshold` attribute specifies for each version trigger the threshold after which the database upgrader gathers performance information from the database. You specify `dbmsperfinfothreshold` in seconds, with a default of 600. If a version trigger takes longer than `dbmsperfinfothreshold` to execute, ClaimCenter:

- queries the underlying database management system (DBMS).
- builds a set of html pages with performance information for the interval in which the version trigger was executing.
- includes those html pages in the upgrader instrumentation for the version trigger.

You can completely turn off the collection of database snapshot instrumentation for version triggers by setting the `dbmsperfinfothreshold` to 0 in `config.xml`.

The `degreeofparallelismforinsertselects` attribute overrides a global default value (2) for the degree of parallelism for all tables involved in insert and select statements in version triggers that benefit from parallelism.

This allows users to take advantage of their hardware when doing an upgrade. This attribute is only applicable for Oracle. Set `degreeofparallelismforinsertselects` to 1 if running on a single CPU machine.

The `<versiontriggers>` element can contain optional `<versiontrigger>` elements for each version trigger. Each `<versiontrigger>` element can contain the following attributes.

Attribute	Type	Description
<code>name</code>	String	The case-insensitive name of a version trigger.
<code>recordcounters</code>	Boolean	Controls whether the DBMS-specific counters are retrieved at the beginning and end of the use of the version trigger. Default is <code>false</code> . If <code>true</code> , then ClaimCenter retrieves the current state of the counters from the underlying DBMS at the beginning of execution of the version trigger. If the execution of the version trigger exceeds the <code>dbmsperfinfothreshold</code> , then ClaimCenter retrieves the current state of the counters at the end of the execution of the version trigger. ClaimCenter writes differences to the DBMS-specific instrumentation pages of the upgrade instrumentation.
<code>extendedquerytracingenabled</code>	Boolean	Oracle only. Controls whether or not to enable extended sql tracing (Oracle event 10046) for the SQL statements that are executed by the version trigger. Default is <code>false</code> . The output can be very useful when debugging certain types of performance problems. Trace files that are generated only exist on the database machine. They are not integrated into the upgrade instrumentation.
<code>queryoptimizertracingenabled</code>	Boolean	Oracle only. Controls whether or not to enable query optimizer tracing (Oracle event 10053) for the SQL statements that are executed by the version trigger. Default is <code>false</code> . The output can be very useful when debugging certain types of performance problems. Trace files that are generated only exist on the database machine. They are not integrated into the upgrade instrumentation.
<code>updatejoinhintbody</code>	String	Oracle only. Used to specify the body of an optimizer hint (without the surrounding <code>/*+ */</code>) for the upgrade of a join in the version trigger. If the update has a default override and the user specifies the empty string <code>""</code> as the value of the attribute, then no hint will be specified.

Starting the Server to Begin Automatic Database Upgrade

The database upgrade is an automatic process that occurs as you start the server with the upgraded configuration. The database upgrade normally completes in a few hours or less.

If the database upgrade stops before completing, then restore your database from the backup, correct any issues reported, and repeat the database upgrade.

IMPORTANT Before starting the upgrade, update database server software and operating systems as needed to meet the installation requirements of the target version. 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>.

WARNING Except for your first database upgrade trials, do not start the server until you have upgraded all rules. Otherwise, default validation rules execute. This could strand objects at a high validation level and make it impossible to edit parts of the object.

WARNING The database upgrade runs a series of version checks prior to making any changes. If any of these checks fail, the upgrade aborts and reports an error message. You can fix the issue, create an updated backup of the database and attempt the upgrade again without restoring from a backup. However, if you experience a failure during the version triggers or upgrade steps portion of the upgrade, refresh the database from a backup before attempting the upgrade again.

Test the Database Upgrade

Prior to attempting the database upgrade on a full-production database clone, test the database upgrade by doing the following:

1. Connected to the built-in H2 database, successfully start the built-in Quickstart (Jetty) application server with a merged configuration data model, including merged extensions, datatypes, fieldvalidators, and so forth.
2. Connected to an empty database on an Oracle or SQL Server database server, successfully start the Quickstart application server from the preceding step.
3. Connected to a restored backup of a production clone, start either the same Quickstart server from the preceding step or a third-party application server with your custom configuration.

Integrations and Starting the Server

Disable all integrations during the automatic database upgrade. Integration points might require updates due to changes in Guidewire APIs. See the *ClaimCenter New and Changed Guide* for specifics.

It is not necessary to have completely migrated integrations before attempting to start the server for the first time. If you have integrations that rely on non-Guidewire applications, do not expect these integrations to work the first time you start the server.

Understanding the Automatic Database Upgrade

As the database upgrade proceeds, it logs messages to the console as well as the log file describing its progress. The database upgrade process requires thousands of steps, divided into three phases. Due to the relational nature of a database, these phases must execute in a specific order for the upgrade to succeed.

During the first phase, the upgrader uses a set of *version triggers* to determine the first actions that need to take place. The database upgrader requires version triggers in order to perform the following types of tasks:

- changing a datatype (other than just length)
- migrating data from one table or column to another
- dropping a column

- dropping a table
- renaming a column
- renaming a table

Specific version triggers are described in this topic.

Many version triggers have version checks associated with them. These checks ensure that the database is ready for the associated version trigger. The database upgrade runs all checks before running any version triggers. If a check detects a problem, it reports the issue, including a sample SQL query to find specific problematic records. If a version check discovers an issue, the database upgrade stops before any version triggers are run. Therefore, it is not necessary to restore the database from a backup if a version check reports an error. Correct the issue and then create a new backup of the database. Then, if you encounter errors after the version check stage, you can restore a version of your database with the issue reported by the version check resolved.

In the second phase, the upgrader compares the target data model and the current database to determine how they differ. The upgrader makes changes to the database that do not require a version trigger during this phase.

Following this process, the third phase runs a subsequent set of version triggers. These triggers create actions that must be run last due to a dependency on an earlier phase.

After the database upgrade concludes, it reports issues that the upgrader encountered and did not complete.

You are responsible for correcting these issues. This might involve modifying the data model or altering the table manually. If you do not correct them, the next time you start the server you do *not* see a message that the database and the data model are out of sync. You must then use the `system_tools` command to verify the database schema.

Note: Given the complexity of database upgrade, Guidewire does not expose specific upgrade actions/steps to clients either in SQL or Java form. Any manual attempts to recreate or control the upgrade process can result in problems in the ClaimCenter database. Recovery from such attempts is not supported.

Version Trigger Descriptions

The database upgrade uses version triggers to perform the actions described by sections within this topic. If a version trigger has an associated version check, the check is described with the trigger. Review these descriptions to familiarize yourself with some of the changes and to understand version checks. If a version check reports an issue, review the error message and consult the description of the relevant version trigger for more information.

Dropping Process History Columns

This step removes the following columns from `cc_ProcessHistory`:

- `PhaseInProgress`
- `PhaseNumber`
- `PhasesToExecute`

Renaming Message Sink Columns

This step renames the `MessageSink` column on the `cc_Message` and `cc_MessageHistory` tables to `DestinationID`.

Dropping Message Sink Tables

This step drops the `cc_loadmsgsinkinsertselect` table and its shadow table `cct_loadmsgsinkinsertselect`.

Creating Subtypes for WorkflowLogEntry

ClaimCenter 6.0 adds three subtypes of the `WorkflowLogEntry` entity: `WorkflowTextLog`, `WorkflowActionLog`, and `WorkflowUserLog`.

This step adds a `cc_workflowlog.subtype` column and sets the value for all rows to the `WorkflowTextLog` subtype.

Upgrading Data Distribution Report Model

This step first removes data distribution reports from the database.

Then, it drops the following tables:

- `cc_revisiondatadist`
- `cc_revisionsizecntdd`
- `cct_revisiondatadist`
- `cct_revisionsizecntdd`

The upgrader then drops the `cc_databasedatadist.RevisionDistsCollected` column, if it exists.

Finally, the upgrader converts the datatype of the following `cc_clobcoldatadist` columns to decimal with a precision of 19 and scale of 0:

- `AverageLength`
- `MaximumLength`
- `MinimumLength`

Dropping Platform Region Type Column

This step removes the `cc_region.regiontype` column. Regions in ClaimCenter 6.0.8 are linked to zones through the new `RegionZone` entity.

Upgrading Holiday Zone Codes

As businesses expand into new geographical areas, there is an increasing chance that a holiday zone code is duplicated. For example, the holiday zone code `CA` could be used to represent California or Canada. To address this issue, ClaimCenter adds `ZoneType` and `Country` properties to `HolidayZone`, enabling you to more precisely define the holiday zone.

The database upgrader first checks that all holiday zone code records have a `cc_holiday_zone.code` value that exists in `cc_zone.code`. If any records are found where `cc_holiday_zone.code` does not exist in `cc_zone.code`, the upgrader logs a message. This message includes a query to find the violating rows. Use this query to identify the rows and then update the rows to use a zone code that does exist in `cc_zone`.

This step then adds and populates two non-nullable typekey columns to `cc_holiday_zone`: `ZoneType` and `Country`. In cases in which `HolidayZone.Code` matches a single `Zone` record, `HolidayZone` takes its `ZoneType` and `Country` values from the `Zone`. In cases in which `HolidayZone.Code` matches multiple `Zone` records, `Holiday.ZoneType` and `Holiday.Country` are populated with the typecode `unknown`. These `HolidayZone` records could not be conclusively populated with `ZoneType` and `Country` values from corresponding `Zone` records. Manually edit these unknown values to real-world values. For example, you could have two `Zone` records with Code `CA`, representing California and Canada. After the upgrade, these `HolidayZone` records have a `ZoneType` and `Country` value of `unknown`. Manually edit these records to set the `ZoneType` and `Country` typekeys.

For more information, see “Columns Added to `HolidayZone` Entity” on page 41 in the *New and Changed Guide*.

Adding Hash of Full Path to SREE Reports

This step adds a `HashOfFullPath` column to `SREEReport` and populates it by calculating a sha1 hash of `SREEReport.FullPath` for each row. ClaimCenter 6.0 calculates the `HashOfFullPath` value each time the `SREEReport.FullPath` column is set. This step removes the uniqueness index from `FullPath` and adds a uniqueness index to `HashOfFullPath`.

Checking Activity Pattern Code Uniqueness

This step checks that no ActivityPattern records have the same Code. If the upgrader finds duplicate activity pattern codes, it writes an error message to the log. This message includes a query to find the duplicate activity codes. Update any duplicate activity pattern codes to unique values before proceeding.

Dropping ExtractReady from Organization Zone Admin

This step removes the ExtractReady column from cc_organizationzoneadmin.

Expanding Geocoding Precision

This step expands the precision defined for the cc_address latitude and longitude decimal columns.

Dropping Columns from Contact Category Score

This step drops the Admin and ExtractReady columns from cc_ContactCategoryScore.

Migrating LodgingProvider

Prior versions included extension columns on PropertyIncident called LodgingNights and LodgingRate. In ClaimCenter 6.0, Guidewire eliminated those columns and added a LodgingProvider array. This step creates cc_LodgingProvider rows where there was data in the two removed columns. If the LodgingProvider entity does not contain LodgingNights and LodgingRate columns, the trigger does nothing. The trigger drops the source columns from PropertyIncident if they are no longer defined as extensions and the trigger successfully created rows in cc_LodgingProvider.

LodgingProvider.LodgingNights exists only for upgrade. In ClaimCenter 6.0 the number of nights is determined using LodgingProvider.StartDate and LodgingProvider.EndDate. If either of those dates is null, ClaimCenter checks LodgingProvider.LodgingNights for data from the upgrade and displays it if present. ClaimCenter never writes to the LodgingProvider.LodgingNights field after upgrade.

Removing instrumentedbatchjobid Column from Work Item Tables

This step removes the instrumentedbatchjobid column from the following work item tables:

- ClaimValidationWorkItem
- ContactAutoSyncWorkItem
- GeocodeWorkItem
- ReviewSyncWorkItem
- WorkflowWorkItem

Dropping Profiler Tables

This step removes the following tables related to the profiler:

- cc_dbProfilerTagPath
- cct_dbProfilerTagPath
- cc_profilerData
- cct_profilerData
- cc_ProfilerPathEdge
- cct_ProfilerPathEdge
- cc_profilerPeriod
- cct_profilerPeriod

Dropping Temporary Contact Tables

This step removes the following tables:

- cc_tempaddressmap

- cc_tempcontactcontact
- cc_tempcontactmap
- cct_tempaddressmap
- cct_tempcontactcontact
- cct_tempcontactmap

Dropping Staging Table Region Type

This step removes the `ccst_region.regiontype` column. Regions in ClaimCenter 6.0.8 are linked to zones through the new `RegionZone` entity.

Dropping Instrumentation Tables

ClaimCenter 6.0 stores batch process performance information in the profiler rather than in instrumentation tables. This step removes the following tables:

- cc_instrumentedjob
- cct_instrumentedjob
- cctt_instrumentedjob
- cc_instrumentedjobappserver
- cct_instrumentedjobappserver
- cc_instrumentedjobparam
- cct_instrumentedjobparam
- cc_instrumentedjobtask
- cct_instrumentedjobtask
- cctt_instrumentedjobtask
- cc_instrumentedjobthread
- cct_instrumentedjobthread

Upgrading Typelist Tables

This step first drops all staging tables. Then for each typelist table, this step expands the size of the typecode column from 22 to 50.

Adding Claim and Policy Currency Columns

This step adds a Currency column to `cc_claim` and `cc_policy` and populates this column with the reporting currency. The reporting currency is the value set to `DefaultApplicationCurrency` in `config.xml`.

See “Policy Currency” on page 26 in the *New and Changed Guide*.

Checking for Negative Values in Non-Negative Incident Columns

A number of columns that were integer datatypes have been updated to only store non-negative integers. This version check reports an error if any of the non-negative columns for Fixed Property, Property or Vehicle incidents contains a negative value. The columns that this version check inspects are listed by incident type:

Fixed Property

- NumSprinkler
- NumSprinkOper
- NumStories

Property

- MealsDays
- MealsPeople
- MealsRate

Vehicle

- OdomRead

Removing Reinsurance Status Typekey and References

This step removes the `ReinsuranceStatus` typekey and references to that typekey. In ClaimCenter 6.0, the reinsurance status feature is replaced by support for reinsurance thresholds. You can preserve reinsurance status data by creating a `ReinsuranceStatus` extension on `Claim` and a `ReinsuranceStatus` typekey. This process is described in “Reinsurance Status” on page 130.

The upgrader first checks if there is data in `cc_claim.ReinsuranceStatus`. If there is data in that column, and no `ReinsuranceStatus` typekey extension defined on `Claim`, the upgrader reports a problem. If you do not want the data, drop the data in `cc_claim.reinsurancestatus` and run the upgrader again. If you do want to keep the data, create a `ReinsuranceStatus` typekey extension on `Claim` and a `ReinsuranceStatus` typelist. If the upgrader detects this extension in the data model, it retains data in `cc_claim.reinsurancestatus` and does not drop the `cctl_reinsurancestatus` and `cctt_reinsurancestatus` tables.

If there is no data in `cc_claim.ReinsuranceStatus`, and there is no `Claim.ReinsuranceStatus` column extension, the upgrader drops the `cc_claim.ReinsuranceStatus` column and the `cctl_reinsurancestatus` and `cctt_reinsurancestatus` tables.

Adding Activity Pattern for Reinsurance Review

This step adds the Claim Reinsurance Review activity pattern.

Adding Claimant Flag on ClaimContact

This step adds a `ClaimantFlag` boolean column to `cc_claimcontact`, if `cc_claimcontact.ClaimantFlag` does not already exist. The upgrader then populates the new column.

For records in `cc_claimcontact` where `cc_claimcontact.ID` equals `cc_claimcontactrole.ClaimContact` and `cc_claimcontactrole.Role` equals `claimant`, and `cc_claimcontactrole.Retired` equals 0, the upgrader sets `cc_claimcontact.ClaimantFlag` to 1. Otherwise, the upgrader sets `cc_claimcontact.ClaimantFlag` to 0.

Adding Transaction and Reporting Amount to Deduction

This step first renames `cc_deduction.Amount` to `cc_deduction.ClaimAmount`.

This step then adds a `cc_deduction.TransactionAmount` column, if the column does not already exist. This column was added in ClaimCenter 5.0.

The upgrader then populates `cc_deduction.TransactionAmount` with the value of `cc_deduction.ClaimAmount`. The upgrader only populates `cc_deduction.TransactionAmount` if it is null.

This step also adds a `cc_deduction.ReportingAmount` column and populates this new column with the value of `cc_deduction.ClaimAmount`.

Adding FixedClaimAmount and FixedReportingAmount to CheckPortion

This step first renames `cc_checkportion.FixedAmount` to `cc_checkportion.FixedTransactionAmount`.

The upgrader then adds a `cc_checkportion.FixedClaimAmount` column, if the column does not already exist. This column was added in ClaimCenter 5.0.

The upgrader then populates `cc_checkportion.FixedClaimAmount` with the value of `cc_checkportion.FixedTransactionAmount`. The upgrader only populates `cc_checkportion.FixedClaimAmount` if it is null.

The upgrader then adds `cc_checkportion.FixedReportingAmount` and populates this new column with the value of `cc_checkportion.FixedClaimAmount`.

Removing Unused Deductible Columns

ClaimCenter 6.0 changes the data model for deductibles. ClaimCenter includes a new `Deductible` entity. Each deductible is associated with a coverage, and by default gets its initial amount from the `Coverage.Deductible` field. A coverage can be associated with zero or one deductible. The `TransactionLineItem` entity includes a foreign key to `Deductible`. The `Deductible` entity is not loadable through staging tables. For details on the deductible data model, see “Configuring Deductibles” on page 633 in the *Configuration Guide*. See also “Deductible Handling” on page 183 in the *Application Guide*.

This step checks that the following columns are empty and not defined as extensions in the data model and then removes them:

- `cc_claim.DeductibleStatus`
- `cc_transaction.DeductibleAmount`
- `cc_transaction.DeductiblePaid`
- `cc_transaction.DeductibleSubtracted`
- `cc_transaction.NetAmount`
- `cc_transaction.OriginalAmount`

If any one of these columns contains data, and is not defined as an extension, the upgrader reports an error, does not remove the data and aborts the upgrade.

If you want to preserve data in any of these columns, define the column as an extension in the data model. If a column is defined as an extension, the upgrader does not remove the column or its data. See “Deductible Fields” on page 129.

Renaming ISO Match Report

This step renames the ISO match report table and accompanying staging and shadow tables, as follows:

Old name	New name
<code>cc_isomatchreport</code>	<code>cc_ExposureISOMatchReport</code>
<code>ccst_isomatchreport</code>	<code>ccst_ExposureISOMatchReport</code>
<code>cct_isomatchreport</code>	<code>cct_ExposureISOMatchReport</code>

Remapping Incidents

This step first checks that:

- for any incident with multiple exposures, the upgraded exposure types agree on the incident subtype.
- for each required incident type change, the new incident type has the same set or a superset of the fields of the old incident.
- for each required incident type change, the new incident type has the same set or a superset of the arrays of the old incident.
- for each required incident type change, any references to the incident are still valid with the new type.

This step then updates `cc_exposure.ExposureType` to match `cc_exposure.CoverageSubtype` if the `CoverageSubtype` to `ExposureType` map has changed.

The upgrader then updates Incident subtypes to match the `ExposureTypes` of referring Exposures.

ClaimCenter 6.0 adds new exposure and incident types for the homeowners policy types. These exposure and incident types are described in the following table, along with their old counterparts for homeowners coverage subtypes.

New Exposure Type	New Incident Type	Old Exposure Type	Old Incident Type	Coverage Subtypes
Dwelling	Dwelling Incident	Property	Fixed Property Incident	<ul style="list-style-type: none"> Dwelling - Property-Damage Earthquake - Dwelling - PropertyDamage Flood - Dwelling - PropertyDamage Mold - Dwelling - PropertyDamage
Living Expenses	Living Expenses Incident	Loss of Use	Property Incident	<ul style="list-style-type: none"> Loss of Use Damage
Other Structure	Other Structure Incident	Property	Fixed Property Incident	<ul style="list-style-type: none"> Other Structure - Property Damage Earthquake - Other Structure - Property-Damage Flood - Other Structure - Property Damage
Content	Property Content Incident	Personal Property	Mobile Property Incident	<ul style="list-style-type: none"> Personal Property Damage Scheduled Property - Personal Property Damage Earthquake - Personal Property Damage Flood Personal Property - Personal Property Damage

The new incidents are subtypes of FixedPropertyIncident.

If you do not want to use any of the new relationships, you can change your CoverageSubtype to ExposureType map using Studio 6.0 before you run the database upgrade.

You can export line of business relationship information from Studio 5.0 and 6.0.

To export line of business relationship information

1. From the command line navigate to ClaimCenter\bin.
2. Enter `gwcc studio` to launch Studio.
3. In the Studio **Resources** pane, click **configuration** → **Lines of Business** → **PolicyType**.
4. In the Navigation panel, right-click **Homeowners** and select **Export Branch** → **HTML** or **Export Branch** → **CSV**.
5. Select a location for the exported HTML or CSV file.
6. Click **Save**.

Adding Activity Patterns for Homeowner Exposures

This step adds the Damaged Items and Living Expenses activity patterns used for the homeowners line of business.

Converting Catastrophe States to Catastrophe Zones

ClaimCenter 6.0 changed from using `CatastropheStates` to `CatastropheZones`. `CatastropheStates` were introduced in ClaimCenter 5.0, so this step only applies if upgrading from ClaimCenter 5.0.x.

You can choose to continue to use `CatastropheState` instead of the `CatastropheZone` functionality. To do so, you create a `CatastropheState` extension table `ccx_catastrophestate`. If the upgrader detects this table in the data model, it renames `cc_catastrophestate` to `ccx_catastrophestate`, preserving data in the table. In this case, this upgrade step stops after the rename operation.

For instructions to preserve `CatastropheState`, see “Retaining Catastrophe States Instead of Catastrophe Zones” on page 127.

If there is no `ccx_catastrophestate` extension, this upgrade step checks that each `cc_catastrophestate.State` code exists as a zone code in `cc_zone` with zone type of state or province. If this check fails, the upgrader logs an error message, including a query to find the catastrophe states that can not be upgraded.

If each `cc_catastrophestate.State` code exists as a zone code in `cc_zone`, the upgrader converts each `CatastropheState` to its `CatastropheZone` equivalent. The upgrader then deletes the `cc_catastrophestate` table and its’ staging and shadow tables, `ccst_catastrophestate` and `cct_catastrophestate`.

The ClaimCenter 5.0.x version of `CatastropheState` included a `Comments` column. If there is any data in this column, and there is a `Comments` column on `CatastropheZone`, the database upgrader copies the data to the `CatastropheZone.Comments` column. The `CatastropheZone` entity does not have a `Comments` column by default. If you have comments in `CatastropheState`, and you want those comments moved to the new `CatastropheZone` entity, create a `Comments` column extension on `CatastropheZone`. Or, if you do not want to preserve the comments from `CatastropheState`, you can alter the `cc_catastrophestate` table to remove the `Comments` column before beginning the upgrade. If the upgrader detects data in the `CatastropheState.Comments` column, and does not detect a `Comments` column extension on `CatastropheZone`, it reports an error message and aborts the upgrade.

For instructions, see “Preserving Comments from Catastrophe State” on page 128.

Adding Currency Column to Authority Limit Profiles

This step adds a `Currency` column to `cc_authorityprofile` and populates the new column with the reporting currency. The reporting currency is the value set to `DefaultApplicationCurrency` in `config.xml`.

See “Currency Aware Authority Limit Profile” on page 30 in the *New and Changed Guide*.

Dropping Source File from Question Set Tables

This steps drops the `SourceFile` column from the following tables:

- `cc_question`
- `cc_questionchoice`
- `cc_questionfilter`
- `cc_questionset`
- `cc_questionsetfilter`

Dropping extractready Column from Certain Tables

This step removes the `extractready` column from the following tables:

- `cc_bulkinvoiceitem`
- `cc_claimassociation`
- `cc_claiminassociation`
- `cc_claiminfo`
- `cc_claiminfoaccess`

- cc_contactinfo
- cc_exposureismatchreport
- cc_isomatchreport
- cc_locationinfo
- cc_message
- cc_periodpolicy
- cc_tmplclaimaccess
- cc_tmpstclaimaccess
- cc_tmpstclaimaccess
- cc_wcbenefitfactordetail

Checking Payment Type

This version check looks for Payments with a null PaymentType. If the version check finds Payments with a null PaymentType, it reports an error. The error message includes a SQL query to identify the Payments with a null PaymentType. You must set a PaymentType for each Payment that this version check reports.

Removing Availability Script from Question Set

This trigger removes the cc_questionset.availabilityscript column. This column was not used by ClaimCenter.

Removing Example Cost Category and Extension Array Entities

This step first checks for any data in the ccx_costcats and ccx_extarray tables. These tables are examples and are not supported for upgrade. If the check finds any records in one of these tables, then it reports an error. If the check passes, this step runs a version trigger to drop the following tables:

- cct_costcats
- ccst_costcats
- ccx_costcats
- cct_extarray
- ccst_extarray
- ccx_extarray

Checking Claim Stubs for Removed Archive States

This version check looks for ClaimInfo records with an ArchiveState of either archiving or markedforarchive. If the version check finds any such records, then it aborts the upgrade and prints a warning message. The database cannot be upgraded when there are items being archived.

Ensuring Claim Loss Locations Are Unique

This version check looks for loss location addresses shared by multiple claims. This version check reports an error if it detects loss location addresses shared by multiple claims. The error message includes an SQL query to identify these claims. If you want multiple claims to share a loss location, update each claim to refer to its own copy of the shared address.

Checking History Records

This version check looks for History rows that have a HistoryType of markedforarchive. The markedforarchive HistoryType is removed in ClaimCenter 6.0. If this version check finds any History rows that have a HistoryType of markedforarchive, it reports an error and provides an SQL query to find the rows. You must manually remove these rows or add the markedforarchive typecode back to perform the upgrade.

Checking for Claims That Were Archived Incorrectly

This version check looks for the following issues with archived claims:

- transferred check exists in the archive database
- Claim referenced by a BulkInvoiceItem
- primary database contains BulkInvoiceItemInfo of an archived Claim.

This version check reports an error if it discovers any of these conditions. Contact Guidewire Support for assistance fixing any issues reported by this version check.

Adding Claim Column to BulkInvoiceItemInfo

This version trigger adds a ClaimID column to cc_biiteminfo. The trigger populates this column based on the ClaimID of the ClaimInfo for the BulkInvoiceItem.

Adding Reporting Columns to Claim and Exposure Report Tables

This version trigger adds and populates the following columns on cc_claimrpt and cc_exposurert:

- AvailableReservesReporting
- FuturePaymentsReporting
- OpenRecoveryReservesReporting
- OpenReservesReporting
- RemainingReservesReporting
- TotalPaymentsReporting
- TotalRecoveriesReporting

The trigger populates the columns with the values contained in the non-reporting versions of these columns. For example, the trigger populates cc_claimrpt.AvailableReservesReporting with the value of cc_claimrpt.AvailableReserves.

Dropping Reporting Tables

This version trigger drops tables for the following entities:

- FinancialsRptUpgrade
- TmpExposureRptStaging
- TmpStagingExposureRpt

After the database upgrade, ClaimCenter creates new tables for these entities upon server startup.

Converting Primary Key Indexes to Clustered Indexes

For SQL Server databases, this version trigger recreates non-clustered backing indexes for primary keys as clustered indexes. This change improves the performance of claim archiving and claim purging operations.

Before recreating the indexes, the version trigger automatically drops (and later rebuilds) any referencing foreign keys and drops any clustered indexes on tables with a primary key.

If you are using filegroups, the upgrade recreates the clustered index in the OP filegroup. By default, the upgrade also stores the intermediate sort results that are used to build the index in the OP filegroup. You can configure the upgrade to instead use the tempdb filegroup for the intermediate sort results.

If you want the upgrade to store the intermediate sort results in the tempdb filegroup, set the sqlserverCreateIndexSortInTempDB attribute of the upgrade element to true.

```
<database ...>
...
<upgrade sqlserverCreateIndexSortInTempDB="true" />
...
</upgrade>
</database>
```

This option increases the amount of temporary disk space that is used to create an index. However, it might reduce the time that is required to create or rebuild an index when tempdb is on a different set of disks from that of the user database.

By default, `sqlserverCreateIndexSortInTempDB` is `false`.

Renaming ClaimInfo Columns

This step renames `cc_ClaimInfo.ArchiveFailure` to `cc_ClaimInfo.ArchiveFailureID` and `cc_ClaimInfo.ArchiveFailureDetails` to `cc_ClaimInfo.ArchiveFailureDetailsID`.

Enforcing Contact Address Uniqueness

This trigger eliminates duplicate Address records in `ContactContact.Address` and `Contact.PrimaryAddress`. For duplicate Address records, the trigger creates a unique Address record and relinks the duplicate references to the unique Address.

Changing Datatype of ArchiveTransitionRec.InternalDesc

This step alters the datatype of the `cc_archivetransitionrec.internaldesc` column from `Text` to `MediumText`.

Changing Precision of Metric Percent Columns

This step alters the precision of the following metric percent columns to eight, with a scale of zero:

- `cc_claimmetric.PercentValue`
- `cc_claimmetriclimit.PercentRedValue`
- `cc_claimmetriclimit.PercentTargetValue`
- `cc_claimmetriclimit.PercentYellowValue`
- `cc_exposuremetric.PercentValue`
- `cc_exposuremetriclimit.PercentRedValue`
- `cc_exposuremetriclimit.PercentTargetValue`
- `cc_exposuremetriclimit.PercentYellowValue`

This trigger only runs if the `cc_claimmetric` table exists already, so it only runs if the database is being upgraded from ClaimCenter 6.0. For database upgrades from earlier versions, the metric percent columns are defined in the data model with the correct precision.

Adding Encryption to Claim Snapshots

ClaimCenter 6.0.8 provides encryption of sensitive data, such as tax IDs, on claim snapshots.

This step adds an `EncryptionVersion` integer column to `cc_claimsnapshot`. As of ClaimCenter 6.0.1, you can create multiple versions of `IEncryption` plugins. The `cc_claimsnapshot.EncryptionVersion` 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 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.

During the upgrade to ClaimCenter 6.0.8, an upgrade trigger increments the global encryption version number. So, when the Encryption Upgrade work queue runs, it will encrypt fields in the snapshot. This trigger does not fire if encryption is not enabled, and there are no encrypted fields defined on the claim snapshot.

For more information about the encryption plugin, see “Encryption Integration” on page 401 in the *Integration Guide*.

For more information about the Encryption Upgrade work queue, see “Batch Processes and Distributed Work Queues” on page 134 in the *System Administration Guide*.

Refactoring Claim and Exposure Staging Tables

This step drops the `ccst_claim` and `ccst_exposure` tables. ClaimCenter 6.0.2 and higher define `Claim.ClaimantDenorm`, `Claim.InsuredDenorm` and `Exposure.ClaimantDenorm` with the attribute `overwrittenInStagingTable` set to true. ClaimCenter automatically populates these columns when you load records into the staging tables. Previous versions of ClaimCenter populated these columns after the staging table records were moved into the main repository. In some cases, this caused performance issues. To remedy this, ClaimCenter 6.0.2 and higher populate these columns when you load the staging table rather than when you move data from the staging tables to the main repository.

Claim and exposure entities are defined as loadable, so ClaimCenter creates new `ccst_claim` and `ccst_exposure` tables when the server starts.

Cleaning up Cross-Claim ReserveLineWrapper.ReserveLine References

The trigger finds instances in which the `BulkInvoiceItemInfo` of a `ReserveLineWrapper` points to a different `Claim` than the `Claim` pointed to by the `ReserveLine`. The trigger updates those `ReserveLineWrapper` entities by repointing a given `ReserveLineWrapper` to a particular `ReserveLine` on its own `Claim`.

The trigger selects the `ReserveLine` belonging to the `Transaction` of the transferred Check. Typically, a `BulkInvoice` Check has only one `Transaction`. However, in this case, cross-claim linkage occurs when the `BulkInvoice` Check is transferred between `Claims`. In that case, the Check has two `Transactions`, an offset on the first `Claim` and an onset on the second. The trigger selects the onset `Transaction` on the original `Claim`.

Correcting Issues with Multiple Exposures on Incidents

If you attempted a database upgrade, and the upgrade reported issues with multiple exposures pointing to the same incident, follow instructions in this topic to resolve the issue. If you did not encounter this error, proceed to the next topic.

It is possible to have more than one exposure point to the same incident. This situation can potentially create a problem during upgrade. The database upgrade uses the `ExposureType` to determine which subtype to make the incident. If these multiple exposures have different `ExposureTypes` (as indicated by the `CoverageSubtype`) that are associated with different `IncidentTypes`, then the upgrade cannot determine the incident subtype. A version check reports this error. See “Remapping Incidents” on page 152.

If you modify the mapping of coverage subtypes to exposure types for ClaimCenter 6.0 to preserve your previous line of business configuration, you will not encounter this issue. The database upgrade will not need to recalculate incident subtypes. You can modify the mapping in Studio.

Otherwise, there are two approaches to handling this issue. The first approach recodes typecodes for each `ExposureType` and `CoverageSubtype` that is associated with an incident with multiple exposures. This affects every reference to the altered `ExposureTypes` and `CoverageSubtypes`, since the change is made in the typelist. The second approach adds a new typecode for each `ExposureType` and `CoverageSubtype` that is associated with an incident with multiple exposures. This typecode is used only for problem exposures. Review both approaches before proceeding.

The first approach only works if the `ExposureTypes` in question are not those with an ID of 11 or below. `ExposureTypes` with an ID of 11 or below are special and cannot be changed. If one of the following `ExposureTypes` needs to be changed, follow Approach 2.

- `BodilyInjuryDamage`
- `LossOfUseDamage`

- PersonalPropertyDamage
- PropertyDamage
- VehicleDamage
- LostWages
- WCInjuryDamage
- EmployerLiability
- GeneralDamage
- PIPDamages
- MedPay

Approach 1

Recode the existing ExposureType and CoverageSubtype typecodes so they do not collide with the new typecodes during upgrade. The general steps are:

1. Recode the typecodes in the pre-upgrade database.
2. Add the recoded typecodes to the target configuration as retired typecodes.
3. Upgrade the database.

After upgrading, existing exposures will have the recoded typecodes but new exposures will have the new typecodes. This might require you to update filters, searches, and any reporting that you are doing against exposures.

Note that this affects every reference to the altered ExposureTypes and CoverageSubtypes, since the change is made in the typelist.

To recode legacy typecodes

1. You tried to upgrade and the upgrade reported the multi-exposure error. This error provides a SQL statement that returns the ids of all Incident rows with multiple Exposures.
2. Determine which ExposureTypes and CoverageSubtypes are causing problems by running the following SQL queries:

```
SELECT DISTINCT exposuretype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

and

```
SELECT DISTINCT coveragesubtype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

3. Change the typecodes in the database. You can do this with two SQL statements, provided by database platform. This example adds the prefix old_ to the typecode:

Oracle:

```
UPDATE cctl_exposuretype
SET typecode = CONCAT('old_' + typecode)
WHERE id IN (<first sql from step 1>)
```

and

```
UPDATE cctl_coveragesubtype
SET typecode = CONCAT('old_' + typecode)
WHERE id IN (<second sql from step 1>)
```

4. Find the affected typecodes from the CoverageSubtype and ExposureType typelist configurations in the old configuration. Copy these into the target configuration.
5. In the target configuration, update the typecode field to match changes you made. Set these typecodes to retired so they do not appear in the user interface when making new exposures, and so forth.
6. Start the ClaimCenter 6.0.8 server to upgrade the database. The exposures that were a problem before do not change during the upgrade. The configuration in the new version for those CoverageSubtypes and ExposureTypes now looks the same as it did in the old version, besides the code field.

Approach 2

Add new typecodes in the pre-upgrade configuration for the problem exposures and then change only those exposures to use the new typecodes. Using this approach, only the problem exposures have an unorthodox ExposureType and CoverageSubtype. All other references to those typecodes will remain.

To recode legacy typecodes only for problem exposures

1. You tried to upgrade and the upgrade reported the multi-exposure error. This error provides a SQL statement that returns the ids of all Incident rows with multiple Exposures.

2. Determine which ExposureTypes and CoverageSubtypes are causing problems by running the following SQL queries:

```
SELECT DISTINCT exposuretype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

and

```
SELECT DISTINCT coveragesubtype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

3. Find the problem ExposureTypes and CoverageSubtypes in the configuration files in the pre-upgrade configuration. Copy each one, giving the copy a new code, such as the existing code prefixed with old_.

4. Start the pre-upgrade server so that the new typecodes are added to the database.

5. Shut down the server.

6. Update the problem Exposure rows so that the CoverageSubtype and ExposureType refer to the new copies. For example:

```
UPDATE cc_exposure
SET exposuretype=<new exposure type id>
WHERE exposuretype=<old exposure type id>
AND incidentid IN (<sql from error>)
```

and

```
UPDATE cc_exposure
SET coveragesubtype=<new coverage subtype id>
WHERE coveragesubtype=<old coverage subtype id>
AND incidentid IN (<sql from error>)
```

Do this for each exposure type and coverage subtype that is causing the problem. Look up the id for the new types in the typelist tables (cctl_exposuretype and cctl_coveragesubtype).

7. Add the new typecodes to the ClaimCenter 6.0.8 target configuration files. You might want to retire them so that ClaimCenter does not display them.

8. Start the ClaimCenter 6.0.8 server to upgrade the database.

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.

Dropping Unused Columns on Oracle

By default, the ClaimCenter database upgrade on Oracle marks some columns as unused rather than dropping the columns. You can configure this behavior by setting the `oracleMarkColumnsUnused` parameter to `false` before running the database upgrade. This parameter is within the `<upgrade>` block of the `<database>` block of `config.xml`. If you set `oracleMarkColumnsUnused` to `false` before the upgrade, the upgrade already dropped removed columns. In that case, you do not need to perform the procedure in this section to drop unused columns.

Marking a column unused is a faster operation than dropping a column. Because these columns are not physically dropped from the database, the space used by these columns is not released immediately to the table and index segments. You can drop the unused columns after the upgrade during off-peak hours to free the space. ClaimCenter does not have to be shutdown to perform this maintenance task. You can drop all unused columns in one procedure, or you can drop unused columns for individual tables.

To drop all unused columns

1. Create the following Oracle procedure to purge all unused columns:

```
DECLARE
  dropstr VARCHAR2(100);
  CURSOR unusedcol IS
    SELECT table_name
    FROM user_unused_col_tabs;
BEGIN
  FOR tabs IN unusedcol LOOP
    dropstr := 'alter table '
              || tabs.table_name
              || ' drop unused columns';
    EXECUTE IMMEDIATE dropstr;
  END LOOP;
END;
```

2. Run the procedure during a period of relatively low activity.

To drop unused columns for a single table

1. Start the server to run the schema verifier. The schema verifier runs each time the server starts. If there are unused columns, the schema verifier reports a difference between the physical database and the data model. The schema verifier reports the name of each table and provides an SQL command to remove unused columns from each table.
2. Run the SQL command provided by the schema verifier. This command has the following format:

```
ALTER TABLE tableName DROP UNUSED COLUMNS
```

Setting Claim Descriptions

ClaimCenter 6.0 provides the option of using claim-level Insurance Services Office (ISO) messaging. Previous versions of ClaimCenter sent exposures to ISO. With ClaimCenter 6.0, you have the option of sending ISO messages at either the claim or exposure level. See “ISO Changes in ClaimCenter 6.0” on page 87 in the *New*

and Changed Guide.

New ISO validation rules require that a description is set on a claim if the claim is to be sent to ISO at the claim level. You will not be able to edit claims that lack a description until you set the description. Any batch processes that modify these claims or their objects, such as activities, will fail until you set the description. The failed batch process will report the following validation error:

```
[java] com.guidewire.pl.system.bundle.validation.EntityValidationException:  
error:entity.Claim.Description/The claim's description must not be null
```

If a claim has already been sent to ISO at the exposure level, then it will never be sent at the claim level. Therefore, the description requirement only applies to claims that have not had any exposures sent to ISO.

If the following conditions are true, check for claims with null descriptions and add a description:

- You have had ISO enabled.
- You want to switch to claim-level ISO messaging.
- You have claims which have null descriptions and have not yet had any of their exposures sent to ISO.

You can check for claims that will cause this validation issue with the following SQL query:

```
SELECT claimnumber FROM cc_claim c  
WHERE c.description IS NULL  
AND NOT EXISTS (SELECT * FROM cc_exposure e  
                WHERE e.ClaimID = c.ID AND e.ISOSendDate IS NOT NULL)
```

If this query returns any claim numbers, set a description for each corresponding claim.

Exporting Administration Data for Testing

Guidewire recommends that you create a small set of administration data from an upgraded data set. Use this data for development and testing of rules and libraries with ClaimCenter 6.0.8. This procedure is optional.

You might have already created an upgraded administration data set by following the procedure “Upgrading Administration Data for Testing” on page 118. If you followed that procedure, or you do not want an administration-only data set for testing purposes, you can skip this topic.

To create an administration data set for testing

1. Export administration data from your upgraded production database.
 - a. Start the ClaimCenter 6.0.8 server by navigating to ClaimCenter/bin and running the following command:
gwcc dev-start
 - b. Open a browser to ClaimCenter 6.0.8.
 - c. Log on as a user with the viewadmin and soapadmin permissions.
 - d. Click the **Administration** tab.
 - e. Choose **Import/Export Data**.
 - f. Select the **Export** tab.
 - g. For **Data to Export**, select **Admin**.
 - h. Click **Export**. Your browser will note that you are opening a file and will prompt you to save or download the file.
 - i. Select to download the admin.xml file.
2. Create a new database account for the development environment on a database management system supported by ClaimCenter 6.0.8. 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 <http://guidewire.custhelp.com>.

See “Configuring the Database” on page 20 in the *Installation Guide* for instructions to configure the database account.

3. Install a new ClaimCenter 6.0.8 development environment. Connect this development environment to the new database account that you created in step 2. See the *ClaimCenter Installation Guide* for instructions.
4. Copy the `admin.xml` file that you exported to a location accessible from the new development environment.
5. Open `admin.xml`.
6. Modify each occurrence of `cc_systemTables` to `systemTables`.
7. Modify the `public-id` of the default organization from `<Organization public-id="default_data:1">` to `<Organization public-id="systemTables:1">`
8. Update all references to the default organization to the new `public-id` by changing each occurrence of `<Organization public-id="default_data:1"/>` to `<Organization public-id="systemTables:1"/>`.
9. Change the root group `public-id` from `<Group public-id="default_data:1">` to `<Group public-id="systemTables:1">`.
10. Change all references to the root group on group users and assignable queues by changing each occurrence of `<Group public-id="default_data:1"/>` to `<Group public-id="systemTables:1"/>`.
11. Change all references to the root group on child groups by changing each occurrence of `<Parent public-id="default_data:1"/>` to `<Parent public-id="systemTables:1"/>`.
12. Save changes to `admin.xml`.
13. Create an empty version of `importfiles.txt` in the `modules/configuration/config/import/gen` directory of the new development environment.
14. Create empty versions of the following CSV files:
 - `activity-patterns.csv`
 - `authority-limits.csv`
 - `reportgroups.csv`
 - `roleprivileges.csv`
 - `rolereportprivileges.csv`Leave `roles.csv` as the original complete file.
15. Import the administration data into the new database:
 - a. Start the ClaimCenter 6.0.8 development server by navigating to `ClaimCenter/bin` and running the following command:
`gwcc dev-start`
 - b. Open a browser to ClaimCenter 6.0.8.
 - c. Log on as a user with the `viewadmin` and `soapadmin` permissions.
 - d. Click the **Administration** tab.
 - e. Choose **Import/Export Data**.
 - f. Select the **Import** tab.
 - g. Click **Browse**.
 - h. Select the `admin.xml` file that you exported from the upgraded production database and modified.
 - i. Click **Open**.

Final Steps After The Database Upgrade is Complete

This section describes the checks and procedures to run after you have completed the upgrade procedure and migration of configurations and integrations. The processes and checks in this section provide you with a benchmark of the new system. Completing these steps is particularly important to going live in a production environment.

IMPORTANT For procedures referenced below that use utilities in `toolkit/bin` on the pre-upgrade configuration, use the utility from `admin/bin` in the target configuration following the upgrade. You do not need to generate the toolkit in the target configuration to use these utilities.

Use these procedures to revalidate the database:

- “Validating the Database Schema” on page 123
- “Checking Database Consistency” on page 123
- “Creating a Data Distribution Report” on page 123
- “Generating Database Statistics” on page 124
- “Running Key Batch Processes and Work Queues” on page 164
- “Backing up the Database After Upgrade” on page 165

Running Key Batch Processes and Work Queues

Before going live, run the following batch processes and work queues while the server is at the MAINTENANCE run level:

- `dashboardstatistics`
- `financialscalculations`
- `claimhealthcalc` (optional)

To set the server run level to maintenance, use the following command:

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

To run a batch process, use the `admin/bin/maintenance_tools` command:

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

These batch processes update the user dashboard, team statistics, financial data and claim metrics in the system.

For a detailed explanation of each batch process, see “Batch Processes and Distributed Work Queues” on page 134 in the *System Administration Guide*.

For more information on the `maintenance_tools` command, see “`maintenance_tools` Command” on page 171 in the *System Administration Guide*.

For a detailed explanation of each batch process, see “Batch Processes and Distributed Work Queues” on page 134 in the *System Administration Guide*.

Calculating Claim Health Metrics

Claim health metrics is a new feature in ClaimCenter 6.0. For more information on this feature, see “Claims Performance Monitoring” on page 26 in the *New and Changed Guide* and “Claim Performance Monitoring” on page 315 in the *Application Guide*.

If you want to use claim metrics, and you want the threshold to apply to existing claims, configure metrics and run the `claimhealthcalc` batch process. If you do not want to use claim metrics, or you do not want metrics to apply for existing claims, you do not need to run the `claimhealthcalc` batch process.

Before you run `claimhealthcalc`, configure claim and exposure metrics for your business. See “Administering Metrics and Thresholds” on page 322 in the *Application Guide*. If you configure metric limits on a development instance of ClaimCenter 6.0.8, import the metric limits into your production environment before you run the `claimhealthcalc` batch process.

To import metric limits from a development environment

1. At a command prompt on the ClaimCenter 6.0.8 development environment, navigate to the `ClaimCenter\bin` directory.
2. Navigate to the `ClaimCenter\bin` directory.
3. Run the command `gwcc dev-start` to launch the ClaimCenter development environment server.
4. Log on to the development environment as a user with the `viewadmin` and `soapadmin` permissions.
5. Click the **Administration** tab.
6. Choose **Import/Export Data**.
7. Select the **Export** tab.
8. Select **Metric Limits** from the **Data to Export** dropdown menu.
9. Click **Export**. Your browser prompts you to open or download the file.
10. Select to download the `metriclimits.xml` file. Save this file in a location where it is accessible from the ClaimCenter 6.0.8 production environment.
11. Launch the ClaimCenter 6.0.8 production environment server.
12. Log on to the development environment as a user with the `viewadmin` and `soapadmin` permissions.
13. Click the **Administration** tab.
14. Choose **Import/Export Data**.
15. Click **Browse**.
16. Select the `metriclimits.xml` file.
17. Click **Finish**.

Upgraded claims and exposures do not have metrics calculated. Run the `claimhealthcalc` batch process to calculate metrics. Some claims and exposures will have metrics that are already yellow or red. For example, one metric tracks how many days a claim has been open. By default, this metric turns yellow after four days and red after six days. The `claimhealthcalc` batch process calculates this and other metrics for each claim. Metric data includes information on when each metric reached yellow and red status. However, after you upgrade and run `claimhealthcalc`, metrics in yellow or red status have a reach time of when the batch process was run. For example, a claim was opened on October 1, is not closed, and you run `claimhealthcalc` on October 10. The claim meets the yellow and red thresholds for the days open metric. The `claimhealthcalc` batch process sets `ClaimMetric.ReachYellowTime` and `ClaimMetric.ReachRedTime` to the time the batch process runs, rather than October 5 and October 7 respectively.

Backing up the Database After Upgrade

Finally, before going live, back up the upgraded database. This provides you with a snapshot of the initial upgraded data set, if an unanticipated event occurs just after going live.

Upgrading Integrations and Gosu from 5.0.x

This topic lists the tasks to upgrade to this release. The tasks are presented in tables, according to when you perform the tasks. You can print these tables to use them as checklists during the upgrade.

This topic includes:

- “Overview of Upgrading Integration Plugins and Code” on page 167
- “Tasks Required Before Starting the Server” on page 169
- “Tasks Required Before Deploying a Production Server” on page 170
- “Tasks Required Before the Next Upgrade” on page 171

Overview of Upgrading Integration Plugins and Code

This topic provides a high level approach to upgrading integration plugins and code. Review this topic, then proceed to the following topics for specific upgrade steps:

- Tasks Required Before Starting the Server
- Tasks Required Before Deploying a Production Server
- Tasks Required Before the Next Upgrade

As part of integration, developers add third-party libraries (JARs) to the toolkit libraries to compile their code. During the upgrade phase, segregate third-party libraries from the toolkit libraries. Initially, it is more practical to make use of these third-party libraries as is during the upgrade process. Later, you can upgrade these libraries separately, along with any ramification to the code, as necessary.

The database upgrade usually matures over the initial cycles of the upgrade process. If the integration code upgrade starts at the same time, regenerating the toolkit might not yield the final version of the toolkit libraries. Consult with the database upgrade team to determine when to regenerate the toolkit for more current libraries.

Integration upgrade steps

1. Create a project with the code at hand after segregating the toolkit libraries from third-party libraries.
2. Ensure you can successfully compile.
3. Create a backup copy of the project.
4. Replace the toolkit libraries with the upgraded toolkit libraries. Leave third-party libraries as is.
5. Update to the correct version of Java. See “Installing Java” on page 27 in the *Installation Guide*.
6. Many classes will fail to compile correctly. The error list is literally the technical upgrade. It needs to be sorted and addressed.

The most commonly encountered compiler failures during upgrade are described in the following table:

Issue	Example	How to approach upgrade
Java upgrades	Refer to Java documentation.	
Changes in object construction	<code>EntityFactory.getEntityFactory().newEntity() → EntityFactory.getInstance().newEntity()</code>	Identify the changes then use a utility to find and replace throughout the code base.
Name changes	<code>gscrip</code> → <code>gosu</code>	Identify the changes then use a utility to find and replace throughout the code base.
Discontinued support of utilities available in previous versions	<code>com.guidewire.util.FileSystem</code> and <code>com.guidewire.util.FileUtil</code>	Carry the old implementation forward as a third-party library.
Class relocation	<code>com.guidewire.logging.SystemOutLogger</code> → <code>gw.util.SystemOutLogger</code>	Locate the new package using searches or a utility such as <code>scanzip</code> in the new toolkit.
Additional interface methods	The <code>IDocumentContentSource</code> interface gained additional methods: <ul style="list-style-type: none"> <code>getDocumentContentsInfoForExternalUse()</code> <code>isInboundAvailable()</code> <code>isOutboundAvailable()</code> 	Review the <i>ClaimCenter New and Changed Guide</i> for additional methods on key interfaces used in your integration plugins.
Functional changes (most involved to upgrade)	Relation between <code>Claim</code> and <code>BodyPartDetails</code> and <code>BodyPartType</code> changing to include <code>InjuryIncident</code> Using a <code>RiskUnit</code> such as <code>VehicleRU</code> or <code>PropertyRU</code> instead of <code>PolicyVehicle</code> or <code>PolicyProperty</code>	Understand the changes and what the code is trying to do and modify your code accordingly. Review database upgrade triggers to understand data model changes. Review the <i>ClaimCenter New and Changed Guide</i>


Tasks Required Before Starting the Server

The following table contains things you must do before you start the server.

 Tasks	For more information...
<input type="checkbox"/> Follow the basic upgrade procedure.	Depending on your starting versions, see one of: "Upgrading from 5.0.x" on page 75 "Upgrading from 4.0.x" on page 173
<input type="checkbox"/> If you modified built-in Gosu enhancements to lists, update your code to merge changes into the new package location	"Moved Enhancement Packages" on page 63 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Check for compile errors due to the new requirement that Gosu blocks cannot contain single statements. To fix this, add braces around the statement. This does not affect expressions as the body of a Gosu block.	"Blocks Require Single Statements in Braces" on page 66 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you wrote any Gosu enhancements on arrays, update the enhancement definition file with the updated syntax.	"Gosu Array Enhancement Changes" on page 72 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency, check for compilation errors in math operations that mix currency amounts and big decimal values. Refactor the code to use currency amounts to preserve the currency information.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove any references to previously-deprecated and now-removed message methods getKey and addKey. Replace with the renamed APIs.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review messaging code and any 'find' queries that reference the message property MessageSink. Update to instead use new name DestinationID.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Check messaging code for compilation errors due to removed message context method for set and get destination and replace with renamed methods.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Check for compilation errors related to package changes of document management plugins and plugin implementations.	"Document Management Plugin Implementation Location Changes" on page 101 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review Java code for compilation errors that require minor change related to stripping parameterization of types in external entities.	"Parameterization of Types Stripped from Java External Entities" on page 101 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you used the validation plugin, refactor to use the Validation rule set instead.	"Validation Plugin Removed" on page 96 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update your development build system to copy the entity libraries JAR file to your configuration module after regenerating it.	"Copy the Plugin Entity Library to Your Configuration Module" on page 86 in the <i>New and Changed Guide</i>
<input type="checkbox"/> For document management storage plugins, add new properties (for Gosu) or methods (for Java) for inbound and outbound method availability.	"Document Management Plugin Changes for Availability" on page 103 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Updating geocoding plugin with a new method signature to get maps with options.	"Geocoding Plugin Changes for Map Options" on page 104 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Fix any compile errors related to changes in block declarations.	"Block Declarations Now Require Argument Names" on page 74 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Fix any compile errors related to changes in function pointers.	"Function Pointers and Nested Functions Now Unsupported" on page 74 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Fix the syntax of any access of annotation data at run time.	"Annotation Syntax Change for Run Time Access" on page 75 in the <i>New and Changed Guide</i>

Tasks Required Before Deploying a Production Server


The following table contains tasks to complete before starting the server and changes to familiarize yourself with before deploying a server to a production environment.

 Tasks	For more information...
<input type="checkbox"/> Review your code for any String literals and Gosu templates that contain <code>\${...}</code> syntax. If you find any, refactor to avoid accidental use of new Gosu String templates, and test your new code.	"Gosu Templates" on page 54 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code for ternary condition expressions (condition ? trueValue : falseValue). Check for any cases in which you pass different types to the true clause and the false clause. If you did not declare the variable type, check for possible changes to the compile-time type of the result.	"Compound Types" on page 62 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review all uses of the run with new bundle API. In the rare case that you use this API in code with a current user, it throws an exception before your block runs. Ensure your code does not rely on the old behavior in this unusual case.	"Exception Changes If No Current User and Creating New Bundle" on page 74 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update any tools or scripts in your build system to use the new locations of scripts and Java libraries.	"Changes to Structure of Integration-related Files and Scripts" on page 85 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Decide whether you want to use new optional claim-based ISO sending. Also, review all ISO code for changes in this release and update your code to use the new payload generation classes. It is possible to use your old payload generation code in this release, but it is best to convert your code to the new system.	"ISO Changes in ClaimCenter 6.0" on page 87 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency and the initial reserves plugin, refactor to use Initial Reserves rule set instead.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency and the policy search plugin, update your plugin to set the currency for a policy.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency, review your code that manipulates currency amounts and big decimal values. Refactor the code to use currency amounts to preserve the currency information.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review code relating to handling duplicate messages and message history. Ensure your code relies on the new behavior.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review messaging code to update method signature for the find by sender reference ID method.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Convert uses of the client-side FNOL mapper to the new server-side FNOL mapper system.	"FNOL Mapper is Now a Server-Side Tool" on page 94 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update any integration code that uses the claim API web service to handle the archiving method name change, and other changes listed in the What's New and Changed book.	"IClaimAPI Web Service Interface Changes" on page 103 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code for use of exposure get recoveries API, and ensure you add the recoveries to a writable bundle.	"Exposure Get Recoveries API Returns Read-only Entities" on page 175 in the <i>New and Changed Guide</i>

Tasks Required Before the Next Upgrade

The following table contains tasks required before the next upgrade. For example, if you used APIs that are now deprecated, begin rewriting your code to avoid use of deprecated APIs. Guidewire will remove these APIs in a future release.

 Tasks	For more information...
<input type="checkbox"/> Review your code related to resource management, streams, and locking. Look for places in which the new using clauses might make code easier to understand or safer.	"Object Lifecycle Management with the 'using' Keyword" on page 56 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code to remove unnecessary downcasting. This makes your Gosu code easier to read and maintain.	"Type Inference Downcasting" on page 56 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code to use the new compact object initializer syntax. This makes your Gosu code easier to read and maintain.	"Object Initializer Syntax During Object Creation" on page 58 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Change any references to gw.api.xml.XMLNode to instead use gw.xml.XMLNode.	"XML Node Package Name Changed" on page 63 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use the old-style XSD registration system (including registry-xsd.xml), update your code to use the new XSD class loading system. This avoids your Gosu code using now-deprecated types.	"Changes to XSD Class Loading Behavior" on page 64 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Change all uses of deprecated collection methods to the renamed versions. Affects: countMatches, findFirst, findAll, findByType, removeMatches, keepMatches. The replacement for findFirst changes behavior with empty sets. While renaming methods, carefully review all code that uses findFirst to ensure you do not rely on the old behavior.	"Changes to Existing Collections Enhancement Methods" on page 67 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code for uses of the now-deprecated list/array expansion operator (the single period). Replace these with the new *. operator syntax.	"New Array/List Expansion Operator (Deprecated Old Style)" on page 71 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove any deprecated variable scoping modifiers and replace with new scoping and concurrency APIs.	"New Concurrency and Scoping APIs, Scoped Variables Deprecated" on page 73 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update any Gosu deprecated references to the plugin registry class. This does not affect Java code.	"New Plugin Registry (Old Version Deprecated in Gosu)" on page 101 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update your code to use custom data model extensions to replace deprecated message properties for message code and optional String data.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review PCF pages, libraries and rules for deprecated usages of Contact.Person and similar casting accessors on the Contact object. Update as necessary.	"Contact Casting Accessors Deprecated" on page 40 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Convert all Gosu template code to use the new template system. If you need to run templates from web services, write custom web services unique to each integration point.	"Gosu Templates" on page 54 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you need to run templates from web services, write custom web services unique to each integration point.	"Gosu Templates" on page 54 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Find deprecated references to type properties and change your code to access those properties directly on the type, not the original object.	"New 'Type' Property on All Types" on page 73 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If any integration code used the template execution web service (the data extraction API), convert these cases to custom web services for each integration point.	"IDataExtractionAPI Web Service Interface Deprecated" on page 102 in the <i>New and Changed Guide</i>

 Tasks	For more information...
<input type="checkbox"/> Update external integration code that uses the messaging tools API method for acknowledging a message, and use the new version.	"IMessagingToolsAPI Web Service Interface Changes" on page 102 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove any uses of the deprecated segmentation plugin and replace with rule set code.	"Segmentation Plugin Deprecated" on page 107 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update financials integration code to use the new domain methods on financials objects, rather than the deprecated message-based methods.	"Financials Integration APIs for Acknowledgement-based Transitions" on page 95 in the <i>New and Changed Guide</i>

Upgrading from 4.0.x

This part describes how to perform an upgrade from ClaimCenter 4.0.x to 6.0.8.

If you are upgrading from ClaimCenter 6.0.x, see “Upgrading from 6.0.x” on page 23 instead.

If you are upgrading from ClaimCenter 5.0.x, see “Upgrading from 5.0.x” on page 75 instead.

This part includes the following topics:

- “Upgrading the ClaimCenter 4.0.x Configuration” on page 175
- “Upgrading the ClaimCenter 4.0.x Database” on page 233
- “Upgrading Integrations and Gosu from 4.0.x” on page 293

Upgrading the ClaimCenter 4.0.x Configuration

This topic describes how to upgrade the ClaimCenter and ContactCenter configuration from version 4.0.x.

If you are upgrading from a 5.0.x version, see “Upgrading the ClaimCenter 5.0.x Configuration” on page 77 instead. If you are upgrading from a 6.0.x version, see “Upgrading the ClaimCenter 6.0.x Configuration” on page 25.

This topic includes:

- “Overview of ContactCenter Upgrade” on page 176
- “Understanding Configuration Modules” on page 176
- “Obtaining Configurations and Tools” on page 176
- “Creating a Configuration Backup” on page 180
- “Updating Infrastructure” on page 181
- “Upgrading the ClaimCenter 4.0 Configuration to 5.0” on page 181
- “Deleting Target Configuration Module” on page 190
- “Remove Size Attribute from Integer and Money Datatypes” on page 190
- “Upgrading the ClaimCenter 5.0 Configuration to 6.0.8” on page 190
- “Upgrading Rules to ClaimCenter 6.0.8” on page 222
- “Importing Translation Properties” on page 223
- “Translating New Display Properties and Typecodes” on page 224
- “Modifying PCF files, Rules and Libraries for Unused Contact Subtypes” on page 225
- “Creating Modal PCF files for Custom Lines of Business” on page 226
- “Converting Velocity Templates to Gosu Templates” on page 227
- “Validating the ClaimCenter 6.0.8 Configuration” on page 229
- “Building and Deploying ClaimCenter 6.0.8” on page 230

- “Integrating ContactCenter with ClaimCenter” on page 231

Overview of ContactCenter Upgrade

The upgrade process for ContactCenter is almost precisely the same as for ClaimCenter. Follow the procedures in this guide to upgrade ContactCenter. Procedures specific to the ContactCenter upgrade are indicated as such. Some information in this guide is specific to ClaimCenter and does not apply to ContactCenter. However, the basic upgrade procedure is the same.

Understanding Configuration Modules

ClaimCenter groups configuration resources in module folders within the `ClaimCenter/modules` directory. 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.

Obtaining Configurations and Tools

Configuration refers to everything related to the application except the database. This includes configuration files such as typelists and PCF files, the file structure, web resources, GScript classes, rules, plugins, libraries, localization files, and application server files.

The upgrade process involves four configurations. This guide refers to these configurations as **base**, **customer**, **intermediate**, and **target**.

Base: The unedited, original configuration on which you based your customer configuration. You can *not* use a copy of this configuration that you might have saved, because the Upgrade Tool requires that the directory structure of the base configuration supports configuration modules. Use the version provided by Guidewire for upgrade purposes specifically. See “Obtaining Configurations” on page 177.

Customer: The configuration you are now using and will upgrade. This is the base configuration of the ClaimCenter version that you currently run with your custom configuration applied.

Intermediate: A ClaimCenter 5.0 installation. You will run the automated steps of the ClaimCenter 5.0 Configuration Upgrade Tool to convert your ClaimCenter 4.0.x customer configuration to ClaimCenter 5.0. You do not need to perform a manual configuration merge to ClaimCenter 5.0. Then you will use the ClaimCenter 6.0.8 Configuration Upgrade Tool to convert the ClaimCenter 5.0 configuration to ClaimCenter 6.0.8. You do not need to upgrade the database to 5.0 before upgrading it to 6.0. You can upgrade the database from 4.0 to 6.0 in one procedure.

Target: The unedited, original configuration of ClaimCenter 6.0.8 on which your upgraded configuration will be based.

Viewing Differences Between Base and Target Releases

To view an inventory of the differences between the base release and the target release, download and carefully review the *Upgrade Diffs Report* from the Guidewire Resource Portal.

1. Open a browser to https://guidewire.custhelp.com/app/projectcenter/upgrades/diff_reports/.
2. Click ClaimCenter or ContactManager.
3. Click **Upgrade From base version**.
4. Click **Upgrade To 6.0.8**.
5. Download the three files into the same folder.
6. Open `Report.html` with a browser.

Obtaining Configurations

Guidewire grants you access to the Guidewire Resource Portal, from which you download the following ZIP files:

- base ClaimCenter 4.0, modified to use the module directory structure
- target ClaimCenter 6.0.8
- intermediate ClaimCenter 5.0

Available base configurations do not include patch releases. If your custom configuration is a patch release, use the base release for which your version is a patch.

Note the `ClaimCenter_Upgrade_Base_Files` and `ContactCenter_Upgrade_Base_Files` directories when downloading the distribution. Within each directory, choose the version of ClaimCenter and ContactCenter that you are upgrading from, such as `cc4.0.8`. Download the ZIP files for ClaimCenter and ContactCenter into local directories. Do not rename the ZIP files and do not unzip them. In a later step, the Configuration Upgrade Tool unzips the base configuration and moves its files into the working directory. Make a note of the location of the ZIP files on your system locally. The `upgrade.properties` file refers to the folder location for the `upgrader.old.base.zip` property.

Unzip the target ClaimCenter 6.0.8 and intermediate ClaimCenter 5.0 into separate directories. Download the latest patch release for the target ClaimCenter 6.0.8 that you are downloading. Follow the instructions with the patch release to install it after you unzip the target version.

IMPORTANT Set all files in the base, customer, intermediate and target configurations to writable before beginning the upgrade.

Specifying Configuration Locations for 5.0 Upgrade Tool

The ClaimCenter 5.0 Configuration Upgrade Tool requires locations of the base and customer environments and a working directory. Define paths to these configurations in the `ClaimCenter/bin/upgrade.properties` file of

the intermediate ClaimCenter 5.0 environment. Remove the pound sign, '#', preceding each property to uncomment the line and then specify values. Use double backslashes in paths. For example, C:\\ClaimCenter. You do not need to use quotes for paths that include spaces.

The following properties are configured in `upgrade.properties`.

Property	Description
<code>upgrader.upgrade.prior.version</code>	The code name for the platform version from which you are upgrading. Enter <code>athena</code> for upgrades from 4.0.
<code>upgrader.old.base.zip</code>	Path to the base configuration .zip file. This is a special ClaimCenter .zip file that uses the modular configuration framework. The ClaimCenter 5.0 Configuration Upgrade Tool unzips this file within the working directory.
<code>upgrader.input.dir</code>	Path to the current customer configuration environment parent directory. This directory contains <code>/bin</code> and other ClaimCenter directories.
<code>upgrader.working.dir</code>	Path to the temporary working directory to be used by the Configuration Upgrade Tool.
<code>upgrader.editor.tool</code>	Path to an executable editing tool. You do not need to use a text editor for the 4.0.x to 5.0 upgrade. However, you can point to one if you want to examine files following the automated steps of the 5.0 upgrade.
<code>upgrader.diff.tool</code>	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool. You do not need to use a merge tool for the 4.0.x to 5.0 upgrade. However, you can point to one if you want to examine files following the automated steps of the 5.0 upgrade.

Property	Description
<code>upgrader.diff.tool.arg.order</code>	<p>The display order, from left to right, for versions of a file viewed in the difference editor tool. By default, the tool displays, left to right, the versions of a file in this order:</p> <p>NewBase PriorBase PriorCustom</p> <p>in which:</p> <p>NewBase is the unmodified target version provided with ClaimCenter 6.0.8.</p> <p>PriorBase is the original base version.</p> <p>PriorCustom is your configured version.</p> <p>The order of these values controls the display order in the difference editor tool. If the tool displays just two versions, it uses the same relative order.</p> <p>By default, the display order places the old base version of a file in the center. The original base version is the common ground between the new uncustomized version and the old customized version. Guidewire changed the old base version to create the new target version, and you changed the old base version to create the customized version in your configuration. With the old base version in the center, you can incorporate both sets of changes to create a customized target version.</p> <p>The default order enables you to merge changes from the left and right to the center and save the merged version. If you use another difference editor, you might need a different order to achieve the same result. Samples are shown below for various difference engines:</p> <p>Araxis</p> <pre>upgrader.merge.tool.arg.order = NewFile OldFile ConfigFile</pre> <p>Beyond Compare</p> <pre>upgrader.merge.tool.arg.order = NewFile ConfigFile OldFile</pre> <p>P4Merge</p> <pre>upgrader.merge.tool.arg.order = OldFile NewFile ConfigFile</pre> <p>You might need to configure the display of your merge tool to show three panels.</p>

Creating a Configuration Backup

Prepare the environment so that you can make a total recovery of the original installation if you run into problems during the upgrade.

Guidewire recommends that you track ClaimCenter configuration changes in a source code control system. Before upgrading, have a labeled version of your entire ClaimCenter installation. A labeled version is a named collection of file revisions.

As an even stronger precaution, make a backup of the same installation directories.

Updating Infrastructure

Before starting the upgrade, have the supported server operating systems, application server and database software, JDK, and client operating systems for the target version. See “Installation Environments Overview” on page 12 in the *Installation Guide* for supported versions for ClaimCenter 6.0.8.

Upgrading the ClaimCenter 4.0 Configuration to 5.0

Use the ClaimCenter 5.0 Configuration Upgrade Tool to upgrade the configuration from 4.0.x to 5.0. You do not need to perform a merge at this step. You only need to run the automated steps of the ClaimCenter 5.0 Configuration Upgrade Tool.

Running the ClaimCenter 5.0 Configuration Upgrade Tool

To launch the ClaimCenter 5.0 Configuration Upgrade Tool

1. Open a command window.
2. Navigate to the /bin directory of the target configuration.
3. Execute the following command:

```
ant -f upgrade.xml merge > upgrade_log.txt
```

You can specify any file to log messages and exceptions. For a typical installation, this command takes a few hours. If it can not complete, restart it.

The Configuration Upgrade Tool performs a number of automated steps. Once the tool completes the automated steps, it opens a user interface. The interface opens whether the automated steps were successful or not.

4. Review the log file or console to verify the automated steps were successful.
5. Close the ClaimCenter 5.0 Configuration Upgrade Tool interface. You do not need to use the ClaimCenter 5.0 Configuration Upgrade Tool interface. You only need to run the tool for the automated steps it performs.

Restarting the Configuration Upgrade Tool

To restart the upgrade, first use the `clean` command to empty the working directories.

```
ant -f upgrade.xml clean
```

ClaimCenter 5.0 Upgrade Tool Automated Steps

The Configuration Upgrade Tool performs a number of automated steps. Review these steps as some might require manual intervention if there is an issue.

Creating Application Modules

Unzips the base configuration into a working directory and a `baseConfig` directory within the working directory. The working directory is specified by the value set to `upgrader.working.dir` in `upgrade.properties`. This is a version of the base configuration that is structured into modules. You obtain this special base configuration from Guidewire. You must list this file, including path, in the `upgrade.properties` file. The file name must be the application code and version number of the base configuration, with the extension `.zip`. This is the `registry.release` value in the customer configuration in `ClaimCenter/bin/registry.properties`.

Creating the Configuration Module

This step creates and populates the configuration module in the working directory. This is the `modules/configuration` directory within the directory you specified with the `upgrader.working.dir` property in `upgrade.properties`. This step populates the working configuration module with files customized in the customer configuration and files present in the customer configuration but not the intermediate configuration.

When this process finds valid customer modifications or additions, it logs that the file is acknowledged and copies the file to the configuration module. If the process finds invalid customer modifications or additions, it logs that the illegal modification or addition is ignored and does not move the file.

This step also logs all files deleted in the customer environment.

Adding Files to Modules

This step copies files into modules. The following table lists changes made to directories and files as they are copied from the customer configuration to the configuration module of the working directory.

Old directory or file	New directory or file in configuration module
<code>config/display/</code>	<code>config/locale/</code>
<code>config/extensions/fieldvalidators_ext.xml</code>	<code>config/extensions/fieldvalidators.xml</code>
<code>config/lib/</code>	<code>lib/</code>
<code>config/javascript</code>	<code>webresources/javascript/global</code>
<code>config/plugins/document/files/templates</code>	<code>config/resources/doctemplates</code>
<code>config/plugins/email/files/templates</code>	<code>config/resources/emailtemplates</code>
<code>config/plugins/note/files/templates</code>	<code>config/resources/notetemplates</code>
<code>config/import/activity-patterns.csv</code>	<code>config/import/gen/activity-patterns.csv</code>
<code>config/import/importfiles.txt</code>	<code>/config/import/gen/importfiles.txt</code>
<code>config/import/roleprivileges.csv</code>	<code>config/import/gen/roleprivileges.csv</code>
<code>config/import/roles.csv</code>	<code>config/import/gen/roles.csv</code>
<code>config/plugins/</code>	<code>plugins/</code>
<code>config/import/authority-limits.csv</code>	<code>config/import/gen/authority-limits.csv</code>

Trim Spaces from Typecode Attributes in Typelists

This step removes leading and trailing spaces from attributes of typecodes in each typelist file.

ClaimCenter 5.0 has more rigid XML validation rules regarding typecode values. Trailing or prefixed spaces are no longer allowed inside a code value.

Deleting Obsolete Directories

This step deletes obsolete directories, if found, from modules in the working directory. The Configuration Upgrade Tool copied these directories, if present, from the customer configuration in a previous step. This step searches for the following directories and removes them if found:

- `config/resources/classes/SmokeTests`
- `config/web/resources`
- `config/web/schema`

Transforming Libraries to Classes

Guidewire has completely removed libraries from its products. This step converts all GScript libraries into GScript classes. Because class names are properly prefixed, all code functions without change or edit. The syntax and name of each old library call and new call to a GScript class are identical.

The upgrade tool takes all files in the existing `config/resources/libraries` directory, then converts the library utilities in this directory and entity extensions in subdirectories to GScript classes or enhancements. You can then find converted classes for utilities in `resources/classes/libraries`, and converted classes for entity extensions in the same directory.

This step logs each library that the upgrade tool converts, and the class or enhancement to which the tool converts the library.

Splitting `displaynames.xml` into Separate Files

The `displaynames.xml` file previously contained entries for many entities. This step divides `displaynames.xml` into multiple files, one for each entity in the original file.

For example, the original file had sections for exposures and claims. This step creates `exposure.xml` and `claim.xml` files.

Renaming Incorrectly Named Typelist Files

This step finds any typelist files for which the name of the typelist file does not match the typelist name. If this process finds any files, it attempts to rename the file to the typelist name. If a file of the same name already exists, this step logs an error.

Merging `localization.xml` and `localization_ext.xml`

This step merges `localization.xml` and `localization_ext.xml`. The `localization.xml` file now contains all changes you previously made in `localization_ext.xml`. You can now edit this single merged file. This step adds this merged file to the `locale` directory in the configuration module of the working directory.

Merging `display.properties` and `display_ext.properties`

This step merges `display.properties` and `display_ext.properties`. The `display.properties` file for each locale now contains all changes and additions made in `display_ext.properties`. You can now edit this one merged file. The `locale/` directory now contains a `display.properties` file for each locale. For example, the files for the United States and for Ottawa are, respectively:

```
locale/en_US/display.properties
locale/fr_CA/display.properties
```

Upgrading the Localization File Schema

This step removes the `<displayProperties>` and `<displaySettings>` subelements from the merged `localization.xml` file. Localized display properties are now stored in a `display.properties` file for each locale.

Localizing Typelists

This step moves localization information for each typecode into a `translation.properties` file. Previously, if you localized the name or description of a typecode, the localization information for the name and desc of the typecode was stored in a locale-specific `display-locale.properties` file. For example, typecodes localized for the French Canadian locale, `fr_CA`, were stored in `/config/display/display-fr_CA.properties`. The upgrader moves the localized typecode information to `/configuration/config/locale/locale/translation.properties`. The upgrader removes the localized typecode information from the `display-locale.properties` file.

This file includes all localized typekeys and display keys. If a typelist is removed in the version you are upgrading to, the upgrader does not move localized typecodes of that typelist to `translation.properties` or remove them from `display-locale.properties`.

In a later manual step, you will import the `translation.properties` file into Studio. See “Importing Translation Properties” on page 223.

Note: Typelist names are case sensitive. Do not change names of existing typelists. The Upgrade Tool does not catch the change, and the subsequent database upgrade will fail.

Deleting Typekey Display Properties

This step removes TypeKey properties from the merged `display.properties` file in each locale.

Copying Display Properties Files into Intermediate Configuration

This step copies `display.properties` files from the `locale` directories of the working configuration module to the intermediate configuration module. The `display.properties` files do not require manual merging, so the Configuration Upgrade Tool copies them directly into the intermediate configuration. If the file already exists in the intermediate configuration, the tool skips the copy and logs a message. This is a precaution to make sure that the Configuration Upgrade Tool does not overwrite customized `display.properties` files if you run the tool again.

Upgrading PCF Syntax

This step upgrades PCF syntax in PCF files in the `config/web/pcf` directory of each module in the working configuration. The Configuration Upgrade Tool makes changes to both base and customer PCF files to ensure compatibility with ClaimCenter 5.0 syntax.

The Configuration Upgrade Tool logs exceptions if it can not make a conversion or if it encounters GScript that it does not understand. The following is a list of syntax changes performed by this upgrade step:

PCF Syntax Changes for ClaimCenter 5.0

- The `mode`, `hideIfEditable`, and `hideIfReadOnly` attributes are removed from `TabBar`.
- The `printer` attribute is removed from all elements.
- The `CommandLineSmokeTest` element is removed.
- The `JavascriptNavigator` element is removed.
- The `FieldDiffRowNode` element is removed.
- The `ArrayDiffRowNode` element is removed.
- The `Differences` element is removed.
- `MessagePayloadInput` is renamed `TextAreaInput` (no attribute changes).
- `InfoBar.Link` is renamed `InfoBarElement`. Its `label` attribute is changed to `tooltip`, and if no `id` attribute is present, the tool generates one using the `value`. Any non-alphanumeric character other than underscore is converted to an underscore when the tool generates the `id` from `value`. If the tool can not generate `id`, most likely because `value` is null, it sets `id` to `NEEDS_ID_#`.
- Any `InfoBarElement` with a null `id` attribute has an `id` attribute generated. The tool generates `id` using the `label`, if `label` is not null. If `label` is null, the tool uses `value`. Any non-alphanumeric character other than underscore is converted to an underscore when the tool generates the `id`. If the tool can not generate `id`, most likely because `label` and `value` are null, it sets `id` to `NEEDS_ID_#`.
- The `icon` attribute on any element has escaped quotes put around it.
- `LockedDateInput` is renamed `Input`, and its `value` attribute is converted to `<previous value>.AccountLocked`.
- `SmokeTestsPage` is renamed `Page` (no attribute changes).
- `BooleanInput` is renamed `BooleanRadioInput`.
- `SelectInput` is renamed `BooleanDropdownInput`.
- `RadioInput` is renamed `RangeRadioInput`.

- `RadioCell` is renamed `RangeRadioCell`.
- The `toolTip` attribute on any element is renamed `tooltip`.
- The `ToolbarFilterOption.default` attribute is renamed `ToolbarFilter.selectOnEnter`.
- Any `conversionExpression`, `toCreateAndAdd`, or `validationExpression` attribute that omitted a return statement has `return null` appended to it. The tool writes a message to the log for each expression to which it appends `return null`. Review these expressions in your PCF files and verify that you do not want a specific value returned.
- `SmokeTestResults` was converted to `Verbatim`, and `id` and `label` attributes were added. The `id` attribute is set to `SmokeTestResults`. The `label` attribute is set to `Form.Message`.
- `SmokeTestTreeViewPanel` is converted to a `TreeView`.
- `WizardStepGroup` is renamed `WizardStepSet`. If the `id` ends with the text `WizardStepGroup`, the tool changes the text to `WizardStepSet`, preserving text that preceded `WizardStepGroup`.
- `SOOMessageSearchItem` is converted to `SearchItem`, and its `ID`, `search`, `onResult`, `onFailure`, `validationError`, and `visible` attributes are set.
- `VerbatimCell` is converted to `Cell` (no attribute changes).
- `Merges` is removed.
- `ProfileStacks` is removed.
- `GroupPickerTreeView` is converted to `TreeView`. Several attributes are added.
- `TeamTreeView` is converted to `TreeView`. Several attributes are added.
- If any element has a `startEditing` attribute set to `true` or `false`, the tool removes the attribute value. The `startEditing` attribute defines an action taken when the page is put in to edit mode. The tool logs a message if it finds any element with a `startEditing` attribute set to `true` or `false`. Check to make sure you did not mean to set the `startInEditMode` attribute instead.
- The `value` attribute on `PickerMenuItem` is converted to `onPick`.
- `Patience` is converted to `DetailViewPanel` containing `Variable` and `InputColumn` with `ProgressInput`. `maxDelay`, `delayCondition`, `onFinished`, `onTimeout` are removed. The attributes `name` and `initialValue` are added on `Variable`. The attributes `id`, `percentage`, and `actionOnComplete` are added on `ProgressInput`.
- The `SampleDataTreeViewPanel` element is removed.
- The `ToolbarFilterOptionGroup.filterSet` attribute value is moved to `filters` and `.FilterOptions` is appended to the value.
- `RegionZipInput`, `RegionCountySelector`, and `RegionStateSelector` are all converted to `TextAreaInput`. This conversion is not a functional conversion. Rather, these widgets could not really be used outside of the base configuration, and the base pages have changed beyond recognition. So this upgrade is just a placeholder, with the expectation that you use or modify the new base region configuration PCF files.
- The `idSuffix` attribute is removed from any element that has one. For any subelements that do not have `id` set, the tool sets the `id` to `item`.
- `DownloadButtonInput` is converted to `ButtonInput`. The `download` attribute is set to `true`.
- `ToolbarDownloadButton` is converted to `ToolbarButton`. The `download` attribute is set to `true`.
- For elements that have a `menuLinks` attribute, if the element is a `Wizard` and `menuLinks` is blank, the tool sets `inheritMenuLinks` to `true`. If the element is not a `Wizard` the tool sets `menuLinks` to `BlankMenuLinks()`.
- For elements with a `tabBar` attribute that is blank, the tool sets `tabBar` to `BlankTabBar()`.
- The `LocationEntryPoint.title` attribute is removed.
- The `MenuTree.width` attribute is removed.
- The `RangeCellType.outputConversion` attribute is removed.
- The `RangeInputBase.outputConversion` attribute is removed.
- The `Note` widget is removed, and the existing PCF changed to use a flexible LV layout.

- BarGraph is converted to:

```
<ChartPanel id="GroupStatisticsBarChart" type="StackedBar" width="800" height="340"
categoryLabelOrientation="Down45">
  <Require name="StatisticsList" type="gw.api.statistics.Statistics[]"/>
  <Require name="Group" type="Group"/>
  <DomainAxis label="" type="Category"/>
  <RangeAxis label="" type="Number"/>
  <DataSeries dataValues="StatisticsList"
    label="displaykey.Java.Desktop.Team.Statistics.Caseload.Tracking"
    categoryLabel="value.Name" value="value.OpenClaims - value.FlaggedClaims"/>
  <DataSeries dataValues="StatisticsList"
    label="displaykey.Java.Desktop.Team.Statistics.Caseload.Flagged"
    categoryLabel="value.Name" value="value.FlaggedClaims"/>
</ChartPanel>
```

- CommandTemplateInput is converted to DocumentTemplateInput, and any value attribute that ended with Command is updated to end with DocumentTemplate.
- The finishConfirmation attribute of NewClaimWizard is set to Wizard.getFinishConfirmation().
- ClaimNumberSearchItem is converted to SearchItem, and search, onResult, onFailure, afterFailure, visible, validationError, and property attributes are added.
- CCTeamTreeView, DashboardTreeView, UserGroupTreeView, and UserPickerTreeViewPanel are converted to TreeView/TreeViewPanel, and several attributes are added.
- ExportDataButton is converted to DownloadButtonInput.
- The info attribute is converted to action.
- The NewTransactionMenu is converted to inline MenuSet.
- Unallowed links are unavailable but visible. They do not show a JavaScript alert like they used to but instead have a tooltip with the original alert message.
- AdditionalPayeePortionInput is converted to InputSet using Choice.
- PolicyStatCodeFilterCriteriaInput is converted to the standard RangeInput.
- ToolbarOptionRangeInput and OptionRangeInput are converted to OptionRangeInput.
- valueRange and optionLabel attributes are added. Child Option elements are removed. A Variable element is added to the nearest containing element which accepts Variable elements. A code block is added to the page.

Address Book PCF Syntax also Includes this Change

- In AddressBook, UserGroupTreeView and UserPickerTreeViewPanel convert to TreeView and TreeViewPanel, and several attributes were added.

The PCF Upgrade Fails in these Unusual Cases

There are a few rare PCF usages that are not handled by the Configuration Upgrade Tool. Correct these usages manually.

Attributes must either return a value (conversionExpression="myExpression(myParameter)") or an expression (conversionExpression="for(item in items) { item.process() }; return items"). However, previous versions of ClaimCenter allowed you to create a conversionExpression, toCreateAndAdd, or validationExpression attribute like conversionExpression="for(item in items) { item.process() }" that did not return anything. The Configuration Upgrade Tool tries to detect this and append return null to such expressions. This might cause problems.

Also, it was possible in the last major version to use Variable elements defined under a RowIterator in the toRemove attribute of the RowIterator. ClaimCenter no longer allows this. In the unlikely event that you have done this, correct the PCF files manually. The Configuration Upgrade Tool does not detect this usage.

Upgrading Plugin Configuration

This step moves plugin configuration elements from `config.xml` to `plugin/plugin-config.xml`. The tool moves the `plugin-container-registry` and `plugin-registry` elements. This step removes the `gwservlet-registry` element, as this element is obsolete.

Upgrading Messaging Configuration

This step moves the messaging configuration element from `config.xml` to `messaging/messaging-config.xml`.

Adding Medium Text Datatype

This step adds the `MediumTextDataType` data type to `fieldvalidators.xml` in the configuration module of the working directory, if there was a custom version of `fieldvalidators.xml` in the customer configuration. The tool sets the length of the `MediumTextDataType` data type to 2000.

Upgrading Contact Role Constraints

This step generates a file titled `entityroleconstraints-config.xml`. This file defines relationships between entities and roles. It defines which Contact subtypes can appear in a role and which contact roles are available to which entities.

The tool extracts information from the customer configuration `tl_cc_claim.xml`, `RoleConstraints.xml` and `ContactRole.xml` files and generates `entityroleconstraints-config.xml` in the working configuration. During the manual merge process, you merge this version with the default version provided in the target configuration. The tool removes the `RoleConstraints.xml` file and edits `ContactRole.xml` to remove all `Category` tags from the codes within the file.

This upgrade step generates a `ContactRoleTypeConstraint` element in `entityroleconstraints-config.xml` for any role for which you had previously defined a subtype in the `ContactRole.xml` `typelist` file. For example, a ClaimCenter 4.0.x `ContactRole.xml` file could include the following:

```
<typecode code="agreedmedexam" name="Agreed Medical Examiner" desc="Agreed Medical Examiner">
  <category typelist="RoleConstraint" code="ClaimEx_Med"/>
  <category typelist="RoleConstraint" code="ExpEx_Med"/>
  <!-- Forbidden for incidents -->
  <category typelist="RoleConstraint" code="IncdProhibited"/>
</typecode>
```

The role `Agreed Medical Examiner` is of subtype `Ex_Medical` on claim and exposure. After upgrade, the `entityroleconstraints-config.xml` file includes the following:

```
<ContactRoleTypeConstraint contactRoleCode="agreedmedexam" contactSubtype="Ex_Medical"/>
```

Your production data might not be able to support the contact subtypes defined in `entityroleconstraints-config.xml`. The following query can be used to determine if a given contact role has multiple contact subtypes in the database:

```
select count(*), CR.typecode as ContactRole, TC.typecode as ContactSubtype
from cc_claimcontactrole CCR, cctl_contactrole CR, cc_claimcontact CC, cctl_contact TC, cc_contact C
where CCR.role = CR.id
and CC.CONTACTID = CCR.CLAIMCONTACTID
and CC.CONTACTID = C.id
and CCR.claimcontactid = CC.id
and C.subtype = TC.id
group by CR.typecode, TC.typecode
order by CR.typecode, TC.typecode
```

If a contact role has multiple contact subtypes, either remove the contact role type constraint or perform a contact subtype conversion so the contacts align to a particular contact subtype. For example, you might find that the `Doctor` contact role has `Person` and `PersonVendor` subtypes in the database. Because of this, you cannot have the following definition:

```
<ContactRoleTypeConstraint contactRoleCode="doctor" contactSubtype="PersonVendor"/>
```

Either remove the above constraint or convert the data such that all doctors are of `PersonVendor` subtype. Otherwise all contacts of type doctor will return a validation error when working with claims created before the upgrade.

Updating the LossPartyType.xml Typelist

This step converts the categorization of `LossPartyType.xml` typecodes from `ExposureType` to `CoverageSubtype`. This conversion is necessary because you can create new `Exposures`, `CoverageSubtypes` and `ExposureTypes`, and you might also classify `LossParty` differently than Guidewire does.

The ClaimCenter data model requires that each `CoverageSubtype` be associated with only one `ExposureType`. Therefore, if the `Exposure` associated with a particular `CoverageSubtype` is categorized as first party, then that `CoverageSubtype` is categorized as first party. Similarly, if the `Exposure` associated with a particular `CoverageSubtype` is categorized as third party, then that `CoverageSubtype` is categorized as third party.

However, since some `ExposureTypes` are categorized in 4.0.x as both first and third party, some `CoverageSubtypes` are categorized as both first and third party by this conversion.

Manually edit the `LossPartyType` typelist to remove unwanted categorizations. For example, `Comprehensive - Vehicle Damage` might be valid for both first and third party after the conversion. Remove one of these two categorizations. In this case, remove the third party categorization.

Merging Incident Typelists

As part of creating incidents, sets of similar typelists used by different LOBs become a single typelist in 5.0. Typically, the generic typelist is preserved and the others are deprecated. For example, `BodyPartType.xml` includes typecodes merged from `WCBodyPartType.xml`. For this automatic merging to succeed, the name attribute of a typecode that occurs in multiple typelists must be identical. If they are not, the Configuration Upgrade Tool inserts a warning comment above the conflicting typecode in the generic typelist file. If this conflict is not resolved, the database upgrade will fail.

The following typelists are affected by this merge process:

- `InjuryType.xml`
- `DetailedInjuryType.xml`
- `BodyPartType.xml`
- `DetailedBodyPartType.xml`
- `MedicalTreatmentType.xml`

During this automated step, if the Configuration Upgrade Tool detects a duplicate in the WC-prefixed version, the tool inserts a warning in the generic typelist. If there is a duplicate, one of the following situations applies:

- You were only using one of the typecodes. This issue does not matter. However, you must modify one of the typecodes so that the name attributes match.
- You were using both typecodes for essentially the same meaning. After the database upgrade, you can update the name or desc field of the generic typecode to reflect nuances in meaning. However, the name attributes must match during the database upgrade.
- The two typecodes must remain separate. Update one typecode to use a different code. This has implications for both the configuration upgrade and database upgrade. Consult Guidewire Support before continuing.

Renaming PCF Files

This step renames certain PCF files in the `shared/contacts` and `addressbook` folders within the `config/web/pcf` folder of the working configuration.

Old name	New name
<code>shared/contacts/AdjudicatorAdditionalInfoInputSet.Adjudicator.pcf</code>	<code>shared/contacts/AdditionalInfoInputSet.Adjudicator.pcf</code>
<code>shared/contacts/PersonAdditionalInfoInputSet.pcf</code>	<code>shared/contacts/AdditionalInfoInputSet.Person.pcf</code>

Old name	New name
shared/contacts/ PersonVendorAdditionalInfoInputSet.PersonVendor.pcf	shared/contacts/ AdditionalInfoInputSet.PersonVendor.pcf
shared/contacts/ UserContactAdditionalInfoInputSet.UserContact.pcf	shared/contacts/ AdditionalInfoInputSet.UserContact.pcf
addressbook/ AddressBookAdjudicatorAdditionalInfoInputSet.Adjudicator.pcf	addressbook/ AddressBookAdditionalInfoInputSet.Adjudicator.pcf
addressbook/ AddressBookPersonAdditionalInfoInputSet.pcf	addressbook/ AddressBookAdditionalInfoInputSet.Person.pcf
addressbook/ AddressBookPersonVendorAdditionalInfoInputSet.PersonVendor.pcf	addressbook/ AddressBookAdditionalInfoInputSet.PersonVendor.pcf
addressbook/ AddressBookUserContactAdditionalInfoInputSet.UserContact.pcf	addressbook/ AddressBookAdditionalInfoInputSet.UserContact.pcf

Populating ISO Destination Settings

This step populates ISO (Insurance Services Organization) configuration files, using settings defined in the customer configuration ISO properties file. By default, this file is `iso.properties`. However, the file to use for ISO properties could be changed by setting `ISOPropertiesFileName` in `config.xml`.

The tool creates `config/plugin/registry/isoTransport.xml` and adds the ISO destination configuration to `config/messaging/messaging-config.xml`.

This step removes the `EnableISOMessageSink` and `ISOPropertiesFileName` parameters from `config.xml`.

Copying Customer Rules to Working Configuration

This step copies rules from the customer configuration `config/resources/rules` directory to the working configuration `config/resources/rules` directory. If this directory already exists in the working configuration, for example if rerunning the tool, this step first deletes all files and folders within before performing the copy.

Fixing Rule Set Types in Rules

This step fixes `rule-set-type` values in rule `.xml` files. The tool first parses each `ruleset.xml` file and collects the `rule-set-type` value. Then the tool checks the `rule-set-type` value of each rule `.xml` file in the rule set directory. If the `rule-set-type` value does not match the `rule-set-type`, the tool updates the rule `.xml` file to specify the `rule-set-type` from `ruleset.xml`.

Reformatting Files

This step adds line breaks to files in the working configuration. This improves file readability and simplifies comparisons with other versions of the files during the merge phase. For example, each attribute of an element now appears on its own line. Additional line breaks are already present in the intermediate configuration and the original version base configuration provided in the `.zip` file.

Copying Customer Rules to Intermediate Configuration

This step copies rules from the customer configuration into the intermediate configuration `config/resources/rules` directory. If the intermediate configuration already contains this directory, this step does not perform the copy and instead logs a message that the directory already exists. This prevents customized rules from being overwritten if rerunning the Configuration Upgrade Tool. If you want to overwrite these rules, delete the `config/resources/rules` directory of the intermediate configuration before running the tool.

Updating ContactCenter Integration Files

If installing both ClaimCenter and ContactCenter, perform these steps during the ClaimCenter upgrade. Do this after the 5.0 Configuration Upgrade Tool completes its automated steps:

1. Create the following directory within the working directory, specified by `upgrader.working.dir` in the ClaimCenter 5.0 `upgrade.properties` file:
`modules/configuration/config/resources/classes/gw/plugin/ccabintegration/impl`
2. Copy the customer versions of `cc-to-ab-data-mapping.xml`, `ab-to-cc-data-mapping.xml`, and `search-filter.xml` from the `GWServices` folder in the 4.0.x base configuration to the new directory.
3. Open `cc-to-ab-data-mapping.xml` and `ab-to-cc-data-mapping.xml` in a text editor.
4. Replace each instance of `cc.plugin.addressbook.mapping.NameTranslatingFieldMapper` in each file with `gw.plugin.integration.mapping.NameTranslatingFieldMapper`.
5. Replace each instance of `cc.plugin.addressbook.mapping.NullFieldMapper` in each file with `gw.plugin.integration.mapping.NullFieldMapper`.
6. Save `cc-to-ab-data-mapping.xml` and `ab-to-cc-data-mapping.xml`.

During the upgrade from 5.0 to 6.0, the ClaimCenter 6.0 Configuration Upgrade Tool detects any changes in these files and presents the proper merge strategy.

Deleting Target Configuration Module

Delete all files and folders in the `modules\configuration` folder of the target ClaimCenter 6.0.8 configuration except `gwmodule.xml`. These files are included with new installations to help rapidly set up a QuickStart environment. You do not need these files to upgrade ClaimCenter, except `gwmodule.xml`, which is required by Studio.

During the manual configuration upgrade process, the Configuration Upgrade Tool copies files that you merge into the configuration module. If you do not delete the contents of the target `modules\configuration` folder, then the Configuration Upgrade Tool locates any file within the folder and considers it merged. However, the file is from the default target configuration rather than an actual merged file.

Remove Size Attribute from Integer and Money Datatypes

Before you run the ClaimCenter 6.0 Configuration Upgrade Tool, check your intermediate configuration `extensions.xml` file for any column extensions of type `integer` or `money`. If you have `integer` or `money` type columns defined, remove any size attribute from the extension column.

The size attribute only applies to `varchar` and `text` type columns. Prior versions of ClaimCenter ignored the error, but ClaimCenter 6.0 reports an error that unknown parameters are specified on the entity when you regenerate the toolkit.

You can fix the issue after the upgrade by removing any `logicalSize` `columnParam` from `integer` and `money` type extensions. However, it is typically simpler to do so before the upgrade while the extensions are contained in a single file.

Upgrading the ClaimCenter 5.0 Configuration to 6.0.8

After you have run the automated steps of the ClaimCenter 5.0 Configuration Upgrade Tool, run the ClaimCenter 6.0.8 Configuration Upgrade Tool to complete the upgrade from 5.0 to 6.0.8. First, specify locations of configurations and tools used to merge the configurations. Then run the ClaimCenter 6.0.8 Configuration Upgrade Tool.

To upgrade your configuration, merge Guidewire changes to the base configuration with your changes. The Configuration Upgrade Tool, provided by Guidewire with the target configuration, facilitates this process.

Specifying Configuration and Tool Locations

The ClaimCenter 6.0.8 Configuration Upgrade Tool depends on the following tools:

- **Text Editor:** An ASCII text editor you use to edit programs and similar files, for example, Notepad, WordPad or Textpad. This editor must not put additional characters in files, as Word does.
- **Merge Tool:** An editor which displays two or three versions of a file, highlights the differences between them, and allows you to perform basic editing functions on them. An example is Araxis Merge. Also known as a “diff tool.” Configure the merge tool to ignore end of line characters to reduce the number of potential false positives during the configuration upgrade step.

The Configuration Upgrade Tool needs the location of the ClaimCenter environment that you will upgrade. The tool stores all versions of files to be merged and merge results in a /tmp directory that it creates within the target environment. Define paths to the configuration and tools in the /ClaimCenter/modules/ant/upgrade.properties file of the target ClaimCenter 6.0.8 environment. Remove the pound sign, '#', preceding each property to uncomment the line and then specify values. Use double backslashes in paths. For example, C:\\ClaimCenter. You do not need to use quotes for paths that include spaces.

The following properties are configurable in upgrade.properties.

Property	Description
upgrader.priorversion.dir	Path to the working directory created by the ClaimCenter 5.0 Configuration Upgrade Tool during upgrade from 4.0.x to 5.0. Use the same value that you specified for upgrader.working.dir in the upgrade from 4.0.x to 5.0. This directory contains /modules and other ClaimCenter directories.
upgrader.editor.tool	Path to an executable editing tool.
upgrader.diff.tool	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool, used for two-way merges. If your difference editor supports both two and three-way merges, you can use the same value for upgrader.diff.tool and upgrader.merge.tool.
upgrader.merge.tool	Path to an executable difference editor tool, such as Araxis Merge Professional or Beyond Compare 3 Professional, also known as a merge tool, used for three-way merges. If your difference editor supports both two and three-way merges, you can use the same value for upgrader.diff.tool and upgrader.merge.tool. You might need to configure the display of your merge tool to show three panels.

Property	Description
<code>upgrader.merge.tool.arg.order</code>	<p>The display order, from left to right, for versions of a file viewed in the difference editor tool specified by <code>upgrader.merge.tool</code>. By default, the tool displays, left to right, the versions of a file in this order:</p> <p>NewFile OldFile ConfigFile</p> <p>in which:</p> <p>NewFile is the unmodified target version provided with ClaimCenter 6.0.8.</p> <p>OldFile is the original base version.</p> <p>ConfigFile is your configured version.</p> <p>The order of these values controls the display order in the difference editor tool. If the tool displays just two versions, it uses the same relative order.</p> <p>By default, the display order places the old base version of a file in the center. The old base version is the common ground between the new uncustomized version and the old customized version. Guidewire changed the old base version to create the new target version, and you changed the old base version to create the customized version in your configuration. With the old base version in the center, you can incorporate both sets of changes to create a customized target version.</p> <p>The default order enables you to merge changes from the left and right to the center and save the merged version. If you use another difference editor, you might need a different order to achieve the same result. Samples are shown below for various difference engines:</p> <p>Araxis Merge Professional</p> <pre>upgrader.merge.tool.arg.order = NewFile OldFile ConfigFile</pre> <p>Beyond Compare 3 Professional</p> <pre>upgrader.merge.tool.arg.order = NewFile ConfigFile OldFile</pre> <p>P4Merge</p> <pre>upgrader.merge.tool.arg.order = OldFile NewFile ConfigFile</pre> <p>You might need to configure the display of your merge tool to show th</p>

Running the Configuration Upgrade Tool

To launch the Configuration Upgrade Tool

1. Open a command window.
2. Navigate to the `modules/ant` directory of the target configuration.
3. Execute the following command:

```
ant -f upgrade.xml upgrade > upgrade_log.txt
```

You can specify any file to log messages and exceptions.

The Configuration Upgrade Tool first copies the modules of the base environment to a `tmp/cfg-upgrade/modules` directory in the target environment. The base environment is specified by the `upgrader.priorversion.dir` property in `ant/modules/upgrade.properties` in the target environment.

The Configuration Upgrade Tool then performs a number of automated steps, described later in this topic. Once the tool completes the automated steps, it opens a user interface. The interface opens whether the automated steps were successful or not. Review the log file or console before proceeding with the manual merge process.

Note: The Configuration Upgrade Tool does not upgrade rules. Merge rules after completing the rest of the configuration upgrade.

Restarting the Configuration Upgrade Tool

The Configuration Upgrade Tool stores work in progress by recording which files you have marked resolved in the `accepted_files.lst` file. This file is stored in the `merge` folder of the target environment. You can close the interface and restart it later without losing your work in progress.

If you do want to start the upgrade over, use the `clean` command to empty the working directories.

```
ant -f upgrade.xml clean
```

WARNING If you empty the `tmp` directory after beginning to merge, you lose all completed merges that you have not resolved and moved into the target configuration directory.

Configuration Upgrade Tool Automated Steps

The Configuration Upgrade Tool prepares for the manual configuration merge process by performing a number of automated steps. Review these steps before proceeding with the configuration merge. Understanding these automated steps helps to understand some file changes you will see when merging the configuration. Finally, some steps might require manual intervention if there is an issue.

Moving Configuration Files

This step moves certain configuration files and directories to new locations. The following table lists old and new locations.

Old location	New location
<code>config/import/security-zones.xml</code>	<code>config/security/security-zones.xml</code>
<code>config/currency/currencies.xml</code>	<code>config/locale/currencies.xml</code>
<code>config/geodata/address-config.xml</code>	<code>config/locale/address-config.xml</code>
<code>config/geodata/zone-config.xml</code>	<code>config/locale/zone-config.xml</code>
<code>config/resources/classes/</code>	<code>gsrc/</code>
<code>config/resources/tests/</code>	<code>gtest/</code>
<code>config/extensions/fieldvalidators.xml</code>	<code>config/fieldvalidators/fieldvalidators.xml</code>

Upgrading Scheduler Configuration

This step upgrades `scheduler-config.xml`. The process names are updated as described in the following table.

Old process name	New process name
<code>abcontactscoring</code>	<code>ABContactScoring</code>
<code>abgeocode</code>	<code>ABGeocode</code>
<code>accountinactivity</code>	<code>AccountInactivity</code>
<code>activityescalation</code>	<code>ActivityEsc</code>
<code>advanceexpiration</code>	<code>AdvanceExpiration</code>
<code>agencysuspendpayment</code>	<code>AgencySuspensePayment</code>
<code>audittask</code>	<code>AuditTask</code>
<code>automaticdisbursement</code>	<code>AutomaticDisbursement</code>

Old process name	New process name
boundpolicyexception	BoundPolicyException
branchsnapshot	BranchSnapshot
bulkinvoicesescalation	BulkInvoiceEsc
bulkinvoicesworkflow	BulkInvoiceWF
chargeproratatx	ChargeProRataTx
claimexception	ClaimException
closedpolicyexception	ClosedPolicyException
collateraleffective	CollEffective
collateralexpiration	CollExpiration
colleffective	CollEffective
collexpiration	CollExpiration
commissionpayable	CmsnPayable
commissionpayment	CommissionPmt
contactautosync	ContactAutoSync
dashboardstatistics	DashboardStatistics
disbursement	Disbursement
excessfund	ExcessFund
exchangerate	ExchangeRate
financialscal	FinancialsCalc
financialsescalation	FinancialsEsc
fullpaydiscount	FullPayDiscount
geocode	Geocode
groupexception	GroupException
idleclaimexception	IdleClaim
invoice	Invoice
invoicedue	InvoiceDue
jobexpire	JobExpire
letterofcredit	LetterOfCredit
newpayment	NewPayment
openpolicyexception	OpenPolicyException
paymentrequest	PaymentRequest
policyrenewalstart	PolicyRenewalStart
premiumreportingreportdue	PremiumReportReportDue
premiumreportreportdue	PremiumReportReportDue
producerpayment	ProducerPayment
receivableaging	ReceivableAging
releasechargeholds	ReleaseChargeHolds
releasettktholdtypes	ReleaseTktHoldTypes
reviewsync	ReviewSync
statementbilled	StatementBilled
statementdue	StatementDue
statistics	Statistics
statreport	StatReport
suitehistorycleanup	SuiteHistoryCleanup
suspensepayment	SuspensePayment
taccountescalation	TAccountEsc
testtimestats	TestTimeStatistics
troubleticketescalation	TroubleTicketEsc

Old process name	New process name
userexception	UserException
userstatistics	UserStats
userstats	UserStats
workflow	Workflow

Moving Work Queue Configuration

This step moves work-queue configuration elements from `config/config.xml` to `config/workqueue/work-queue.xml`. This step also replaces the name attribute of the work-queue element with `workQueueClass`, according to the following table.

Work queue name	Work queue class
ABGeocode	<code>com.guidewire.ab.domain.geodata.geocode.ABGeocodeWorkQueue</code>
ABContactScoring	<code>com.guidewire.ab.domain.contact.ABContactScoringWorkQueue</code>
ClaimValidation	<code>com.guidewire.cc.domain.claim.impl.ClaimValidationWorkQueue</code>
ContactAutoSync	<code>com.guidewire.pl.system.contactautosync.ContactAutoSyncWorkQueue</code>
Geocode	<code>com.guidewire.pl.domain.geodata.geocode.GeocodeWorkQueue</code>
ReviewSync	<code>com.guidewire.cc.domain.contact.ReviewSyncWorkQueue</code>
Workflow	<code>com.guidewire.pl.system.workflow.engine.monitor.WorkflowDistributedWorkQueue</code>

Upgrading Data Model and Typelist Metadata Files

This step converts entities in data model and typelist metadata files in the `config/metadata` folder of each module to the new format. Each entity element in these files becomes a separate file. The file extension of the new file depends on the entity type.

File extension	Entity types
.eix	<code>internalExtension</code>
.eti	<code>component</code> <code>delegate</code> <code>entity</code> <code>extensionarray</code> <code>nonPersistentEntity</code> <code>subtype</code> <code>viewEntity</code>
.etx	<code>extension</code> <code>viewEntityExtension</code>
.tix	<code>internaltypelistextension</code>
.tti	<code>typelist</code>
.ttx	<code>typelistextension</code>

Data model metadata files were previously named using the format `dm_cc_name.xml`. They now use the format `name.eti`. Data model extension files use the name format `name.etx`. Internal data model extensions use the name format `name.eix`.

Typelist metadata files were previously named using the format `tl_cc_name.xml`. They now use the format `name.tti`. Typelist extension files use the format `name.ttx`. Internal typelist extensions use the name format `name.tix`.

This step also converts `extensions.xml` into separate `entityName.etx` files for each extension. ClaimCenter 6.0 no longer uses the `extensions.xml` file for extension and custom entity definitions. For information on creating entities and extensions to the ClaimCenter data model, see “Modifying the Base Data Model” on

page 233 in the *Configuration Guide*.

IMPORTANT This step copies comments from above entity or extension definitions in `extensions.xml`. The upgrade copies everything after the prior `</extension>` or `</entity>` closing tag to the new file for the entity or extension. If you have comments in `extensions.xml`, make sure the comments are placed above or within the entity or extension definition. Then, the upgrade copies the comments to the correct new file.

This step also makes the following changes to the metadata:

- Replaces `<extensionarray>` elements with the basic `<array>` element. The `extensionarrayentity` attribute on these elements has been changed to `arrayentity`. The `extensionarraytable` attribute has been removed from these elements.
- Replaces `<extensionarraytable>` elements with an `<entity>` of type `joinarray`.
- Removes the `table` attribute from `<nonPersistentEntity>`.
- Changes the `long` column type to `longint`.
- Changes the root element name of typelists that are extensions from `<typelist>` to `<typelistextension>`. Removes the `final` attribute from such typelists. Removes the `abstract` and `baseList` attributes.
- Fixes references to base entities. For example, the `Activity` entity used to have a base entity, `ActivityBase`. Prior to ClaimCenter 6.0, you could refer to `Activity` either as `Activity` or `ActivityBase`. Now you can only refer to `Activity` as `Activity`. The upgrader changes any reference to `EntityBase` to `Entity`. For example, the upgrader changes any reference to `ActivityBase` to `Activity`.
- The default for the `setterScriptability` attribute was set to `all` for most fields, but `doesNotExist` for `<onetoone>` elements. Now, it has the same default, `all`, for all fields. The upgrader sets `setterScriptability` to `doesNotExist` for any `<onetoone>` element without an explicit `setterScriptability` attribute defined.
- The `<validationTriggerOverrides>` element has been removed. Instead, these overrides are handled as field override elements. If a `<validationTriggerOverride>` existed for a field, the upgrader replaces it with either a `<foreignkey-override>`, `<array-override>`, or `<onetoone-override>` element, depending on the type of field being overridden. The upgrader adds these overrides to the extension for the owning entity. If no extension exists for that entity, the upgrader creates a new one.
- Replaces the `<delegatesto>` element with `<implementsEntity>` or `<implementsInterface>`, depending on whether the type of delegate being implemented is an entity or a basic java object.
- Removes `<arg>` elements appearing under `<delegatesto>` elements. These were never used and are obsolete.
- Removes the `implements` attribute and replaces it by splitting up the value (comma-separated) and creating `<implementsEntity>` elements for each one.
- Changes foreign key names ending in "ID" by removing the "ID" suffix from the name, and setting the `columnName` attribute to the former name. Fixes references to the old name in the following:
 - the `arrayfield` attribute of `<array>`
 - the `linkField` attribute of `<onetoone>`
 - the `fieldName` attribute of `<validationTriggerOverride>`
 - the `path` attribute of `<viewEntityColumn>`, `<viewEntityName>`, `<viewEntityTypekey>`, and `<viewEntityLink>`
 - the `paths` attribute of `<computedcolumn>` and `<computedtypekey>`
- Removes the obsolete `immutable` attribute from `<foreignkey>`, `<typekey>`, and `<array>`.

Upgrading Column Attributes to Subelements

This step upgrades certain attributes of column entities into separate columnParam elements. This step operates on .eti and .etx files created by the last step in the config/metadata and config/extensions folders. The following attributes are converted to columnParam elements:

- size
- scale
- precision
- encryption
- trimwhitespace

The type attribute and other attributes remain on the column.

For example, the entity:

```
<entity entity="TestOverride" ...>
  <column name="ScaleOverride" precision="5" scale="2" type="decimal"/>
</entity>
```

becomes:

```
<entity entity="TestOverride" ...>
  <column name="ScaleOverride" type="decimal"/>
    <columnParam name="precision" value="5"/>
    <columnParam name="scale" value="2"/>
  </column>
</entity>
```

This step changes the size attribute for all column types other than varchar to logicalSize.

If the column type is encryption, this step changes the type to varchar and adds the following columnParam element:

```
<columnParam name="encryption" value="true"/>
```

Deleting Unused Extensions

This step parses the extension .eti and .etx files created in a previous step and adds the following to the customer extensions directory:

- A <deleteEntity> for any extension entities that were removed.
- Empty <extension> for any entity extensions that were removed. These are extensions that have been removed from entities that remain.
- Empty <viewEntityExtension> for any view entity extensions that were removed.

IMPORTANT If you had deleted an out of the box Contact subtype, such as Adjudicator, check addressbook/contact-sync-config.xml for any references to that entity that will no longer be used. Remove any references to the deleted Contact subtype in contact-sync-config.xml.

Deleting Old Format Metadata Files

This step deletes the old metadata files dm_*, tl_*, extensions.xml, typelists/*.xml, metadataproperties.xml, and platform_metadataproperties.xml from the working directory.

Upgrading Field Validator Configuration

This step moves DataTypes elements from /config/fieldvalidators/fieldvalidators.xml to a separate file: /config/fieldvalidators/datatypes.xml. An earlier step moved fieldvalidators.xml from /config/extensions to /config/fieldvalidators. This step also creates the /config/fieldvalidators/fieldvalidatorcountries directory for country-specific field validator overrides and ensures the proper schema location.

Replacing Field Validator Override and Field Length Override Elements

This step removes `FieldValidatorOverride` and `FieldLengthOverride` elements from the `fieldvalidators.xml` file.

The tool converts `FieldValidatorOverride` elements to `column-override` elements on the extension itself. For example, the default `fieldvalidators.xml` file in ClaimCenter 5.0 contained the following `FieldValidatorOverride` on the `Contact` object:

```
<FieldValidatorOverride
  entityname="Contact"
  fieldname="EmailAddress1"
  validatorname="Email"/>
```

The upgrade tool adds the following to the `extensions/Contact.etc` file:

```
<column-override name="EmailAddress1">
  <columnParam name="validator" value="Email"/>
</column-override>
```

The upgrade tool converts `FieldLengthOverride` elements to `logicalSize` elements on the extension.

Setting XML Namespace on Metadata Files

This step sets the XML namespace on data model and typelist entity and extension files in `config/metadata` and `config/extensions` to `http://guidewire.com/datamodel` and `http://guidewire.com/typelists` respectively. You can configure an XML editor to map these namespaces to XSD files that define the structure of data model and typelist files. Map `http://guidewire.com/datamodel` to `ClaimCenter\modules\pl\xsd\metadata\datamodel.xsd` and `http://guidewire.com/typelists` to `ClaimCenter\modules\pl\xsd\metadata\typelists.xsd`. Then, the XML editor can validate entities as you create or modify them.

Moving Typelist Localizations into Translation Properties Files

This step moves typelist localization elements into `translation.properties` files, in order to facilitate typelist merging. This step creates a `translation.properties` file for each locale containing the localized name and description of each localized typecode.

Upgrading PCF Files

This step upgrades PCF files from ClaimCenter 5.0.x format to ClaimCenter 6.0.8 format.

- The `afterCommit` attribute of popup page type PCF files is modified to replace `CurrentLocation` with `TopLocation`. `TopLocation` always represents the top location on a stack.
- `FinancialsSummaryLV.pcf` is updated as follows:
 - `FinancialsSummaryCell` is replaced with a `TextCell`. The `ColumnHeader` subelement value becomes a `label` attribute.
 - `FinancialsSummaryRow` is replaced with a `Row`.
 - `FinancialsSummaryLabel` name and value attributes are updated to match the new version of `FinancialsSummaryLV.pcf`, but otherwise stay the same.

The upgraded `FinancialsSummaryLV.pcf` does not exactly match the new version. Some simple merging is still required during the configuration merge step.

Upgrading Rules to Gosu Classes

This step upgrades rules to the new Gosu class format.

Rule Sets

The upgrader converts the old XML file format into a directory structure that represents the hierarchy of the rule set.

The top-level conversion converts the rule set into:

- a rule set Gosu class.
- a directory to contain all of the child rules. The directory is named using the convention *ruleset_dir*.

The rule set Gosu class contains the following upgraded rule set info:

- name (in the `RuleSetName` annotation)
- description
- type name (the Gosu class)

The directory contains the following:

- an `order.txt` file (representing the rules under this rule set)
- other rules for this class

The order of the top level rules (formerly in `ruleset.xml`) is now in `order.txt` under the rule set directory. The `order.txt` file contains relative type names for the rules to run. Individual rules can be disabled and that is represented by an annotation described below.

The rule set type is now encoded in the package name of the rule set. The code was changed to not use the type but instead use a readable package name.

The root bean object is encoded in the name of the type for the Preupdate and Validation rules.

Rules

Individual rules follow the same directory structure as rule sets. There is a primary Gosu class, a subdirectory (name of rule + "_dir") and an `order.txt` file.

The rule set Gosu class contains the following upgraded data:

- rule set name (in the `RuleName` annotation)
- condition (converted into the `doCondition` method)
- action (converted into the `doAction` method)

The `messageContext` property used in the condition and action is converted as arguments to the `doCondition` and `doAction` methods. Furthermore the actions are converted into the `actions` parameter of the `doAction` method.

The action and condition are wrapped in the special tokens `/*start00rule/` and `/*end00rule*/`. These must be present or Studio cannot parse the file.

The order of subrules is now in the `order.txt` file along with relative type names for the rules.

Subrules under a rule follow the same process, recursively.

IMPORTANT You cannot perform a merge on rules because of this format change. Instead, after merging the rest of the configuration, use Studio to compare the ClaimCenter 6.0.8 default rules with your rules and update as necessary.

Adding Default Rules

This step moves the default rules provided with ClaimCenter 6.0.8 to a single folder. Because the configuration upgrade converts rules from XML to Gosu, you cannot perform a merge of rules as part of the upgrade. Instead, use Studio to compare the default rules with your custom rules and make updates.

Updating Plugin Template Suffixes

This step updates extension suffixes of plugin template files in `\config\templates\plugins` of each module from `.gs` to `.gsm`. Some older plugin interfaces use text generated by these templates to pass parameters from ClaimCenter to a Java or Gosu plugin implementation.

Removing AdminTable Delegate from Custom Extensions

This step removes the `AdminTable` delegate from custom extensions.

Upgrading ISO Coverage Mapping

This step produces mappings for each policy type mapped in `iso/TypeCodeMap.xml`. For each policy type, it finds all coverage types and coverage subtypes by using the categories in the typelist files as set up by the LOB editor. It then builds a mapping for each valid `policy type/coverage type/coverage subtype` triplet. If `TypeCodeMap.xml` does not contain a mapping for any one of the types (`policy`, `coverage` or `coverage subtype`), then the upgrader omits that triplet.

The upgrader finds applicable coverage types and coverage subtypes using case-insensitive matching. This is because typelist files are case-insensitive with respect to typecodes. However, the upgrader creates a new file, `ISOCoverageCodeMap.csv`, using the exact case given in the original `TypeCodeMap.xml`. For example, if typecode `abc` was sometimes listed as `ABC` in typelist categories but was listed as `ABC` in `TypeCodeMap.xml`, then it will appear as `ABC` in `ISOCoverageCodeMap.csv`.

The upgrader does not modify the `TypeCodeMap.xml` file, or any of the typelist files. If you will continue to use the old payload generation code, you will still need the original `TypeCodeMap.xml`.

This upgrade step is intended to produce a first draft of the `ISOCoverageCodeMap.csv` file. It does not take into account any custom configuration of the `policy type/coverage type/coverage subtype` that is done in the Gosu messaging code. Manually modify `ISOCoverageCodeMap.csv` to take account of such customizations.

Renaming Exchange Rate PCF Files

This step renames the two modes of `ExchangeRateInputSet.pcf` to be two separate files. These files are in the `config/web/pcf/claim/shared/` directory. The upgrader renames `ExchangeRateInputSet.Check.pcf` to `CheckExchangeRateInputSet.pcf` and `ExchangeRateInputSet.default.pcf` to `TransactionExchangeRateInputSet.pcf`.

Updating PCF Files to Reference Namespace and XML Schema

This step updates PCF files to reference the XML schema instance namespace and schema. The schema is provided with ClaimCenter 6.0 in `ClaimCenter\modules\pcf.xsd`. The configuration upgrade sets the schema location for each PCF file to point to the relative location of `pcf.xsd`. For example, the configuration upgrade adds the following to `ClaimCenter\modules\cc\config\web\pcf\admin\admin.pcf`:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../pcf.xsd">
```

Copying Display Properties Files into Target Configuration

This step copies `display.properties` files from the `locale` directories of the working configuration module to the target configuration module. The `display.properties` files do not require manual merging, so the Configuration Upgrade Tool copies them directly into the target configuration. If the file already exists in the target configuration, the tool skips the copy and logs a message. This is a precaution to make sure that the Configuration Upgrade Tool does not overwrite customized `display.properties` files if you run the tool again.

Reformatting Files

The Configuration Upgrade Tool performs a number of steps to reformat files so they are more easily merged during the manual merge process.

The tool sorts the content of typelist files in `/config/metadata` and `/config/extensions` by typecode to simplify the manual merge process.

The tool formats XML elements in files in the following directories to place tags on separate lines:

- `/config/metadata`
- `/config/extensions`
- `/config/web/pcf` and subfolders
- `/config/displaynames`
- `/config/workflow`
- `/config/locale/localization.xml`
- `/config/plugin/registry`

The tool removes carriage return characters from ASCII files, converting these files to UNIX format.

Specifying XML Namespaces

This step adds a namespace specification to certain XML files.

XML file	Namespace
<code>config/config.xml</code>	<code>http://guidewire.com/config</code>
<code>config/currency/currencies.xml</code>	<code>http://guidewire.com/currencies</code>
<code>config/locale/localization.xml</code>	<code>http://guidewire.com/localization</code>
<code>config/messaging/messaging-config.xml</code>	<code>http://guidewire.com/messaging-config</code>

Moving XSD Files to gsrc Directory

This step moves configuration module XSD files that are listed in `config/registry/registry-xsd.xml` from `configuration/config/registry/xsds` to `configuration/gsrc/xsd`. The Configuration Upgrade Tool uses the namespace listed for the XSD within `registry-xsd.xml` to create the new file name at the new location.

Using the ClaimCenter 6.0.8 Upgrade Tool Interface

IMPORTANT Review the automated step descriptions before you proceed. Some automated steps might require you to perform a manual step while merging the configuration. Typically, such automated steps insert a warning into a file. Check the `steps_results.txt` file for warning and error messages. Correct any issues reported. Then, delete `steps_results.txt` and restart the Configuration Upgrade Tool.

After the Configuration Upgrade Tool completes the automated steps, the working area contains up to three versions of the same file:

- the *customer* file
- the *base* file, from which you configured the customer file
- the *target* file, from ClaimCenter 6.0.8.

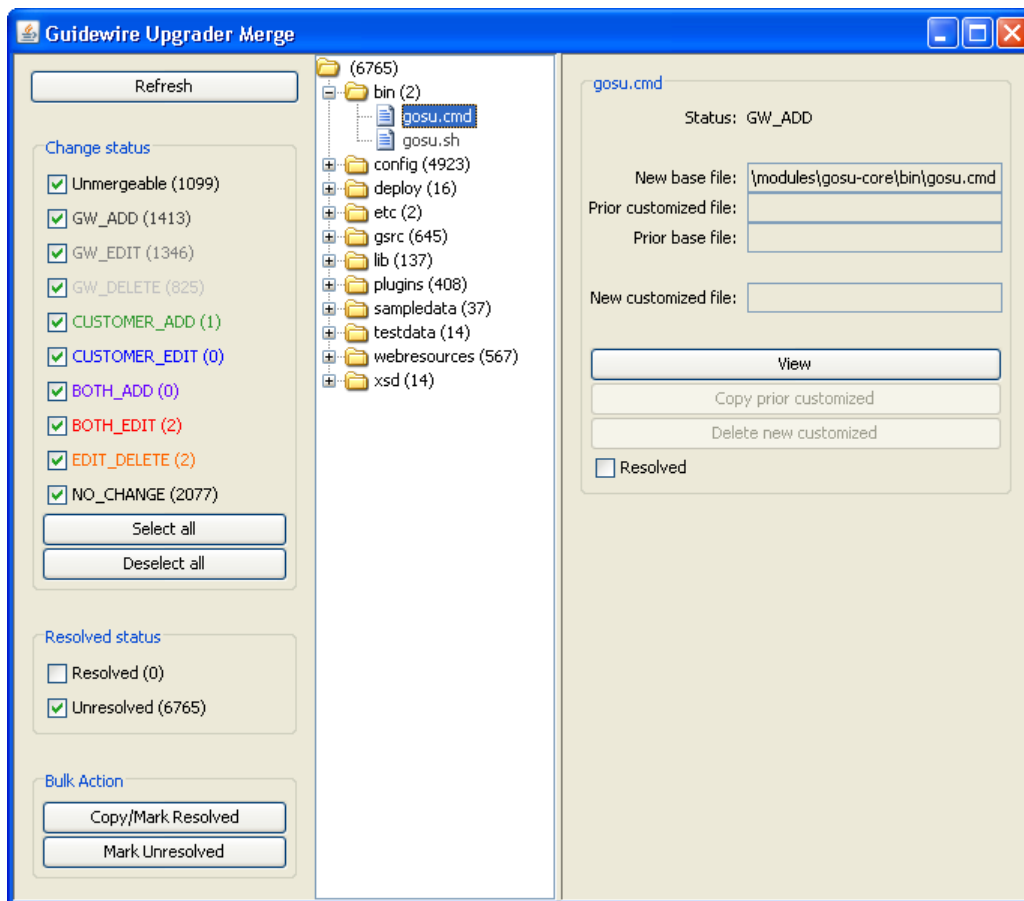
In the manual process of the upgrade, you decide whether to use one of these versions unchanged, or merge versions together. The Configuration Upgrade Tool provides a user interface to assist with the manual process. This interface has several important functions:

- It shows a complete list of all configuration files, organized into the module directory structure.

- It allows you to filter this list. You can, for example, view a list of all files that differ between the target version and your version. See “Change Status Filters” on page 203.
- It displays two or three versions of a file and their differences, using a merge tool you supply, such as Araxis Merge or P4Merge, defined in `upgrade.properties`.
- It lets you edit your file, incorporating changes from the other file versions, and save it.
- It lets you accept this merged version instead of one of the previous versions.
- It lets you edit the file after you have accepted changes from the merge using the text editor defined in `upgrade.properties`.

After you have accepted or merged all files that the Configuration Upgrade Tool displays, the merging process is complete.

The Configuration Upgrade Tool displays three panels. The center panel is a tree view of the files in the configuration, filtered by filter choices selected in the left panel. Files appear in the color of the filter that found them. As you select a file in the center panel, the right panel displays file information and buttons to perform actions on that file.



Filters

The left panel of the Configuration Upgrade Tool contains:

- Refresh Button
- Change Status Filters
- Resolved Status Checkboxes
- Bulk Action Buttons

Refresh Button

If multiple users are working in the same directory, each user can mark files as resolved. The **Refresh** button refreshes the resolved status of files shown in the Configuration Upgrade Tool for changes contributed by all users working in the same directory.

Change Status Filters

This table lists the change status filters that the Configuration Upgrade Tool displays in the left panel. Use the check boxes next to the filters to select one or any combination of change statuses to view. Use the **Select all** or **Deselect all** buttons to select or deselect all filters. The following table describes change status filters. The Guidewire Action column lists the change Guidewire has made to files matching a status filter since the prior version. The Your Action column lists the change to the file in your implementation:

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
Unmergeable	change format of file	any	file exists in a different format and thus cannot be merged with an old version	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file, in the new format, already exists in the target configuration.</p> <p>The Configuration Upgrade Tool automatically marks certain files as unmergeable, including rules and display properties files. The Configuration Upgrade Tool upgrades these files before the interface displays.</p> <p>You can also specify a regular expression pattern in <code>upgrade.properties</code> for file paths to mark files matching that pattern as unmergeable. Set the pattern as the value of the <code>exclude.pattern</code> property.</p> <p>Typically, you use <code>exclude.pattern</code> to specify source control metadata files. Samples are provided in <code>upgrade.properties</code> for CVS and SVN.</p>
GW_ADD	add	none	file in target not in base	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file added by Guidewire already exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
GW_EDIT	edit	none	file in target differs from base	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file added by Guidewire already exists in the target configuration.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a comparison between the new Guidewire version and the original base version. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
GW_DELETE	delete	none	file in base not in target	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file deleted by Guidewire no longer exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
CUSTOMER_ADD	none	add	file in customer configuration only	<p>If you resolve the file, the Configuration Upgrade Tool copies the file to the target configuration module if the file has not been copied there already.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
CUSTOMER_EDIT	none	edit	<p>file differs between customer and base configurations</p> <p>file unchanged between base and target configurations</p>	<p>If you resolve the file, the Configuration Upgrade Tool copies the file to the target configuration module if the file has not been copied there already.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a comparison between your custom version and the original base version. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>
BOTH_ADD	add	add	new file with matching name in both target and customer configurations (rare)	<p>You must either merge the two versions of the file or copy your prior version of the file into the configuration module before you can resolve the file.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code> to perform a merge between your version and the Guidewire version. If you make changes, the tool prompts you to copy the merged file to the target configuration module.</p>
BOTH_EDIT	edit	edit	file changed in both customer and target configurations	<p>You must either merge the two versions of the file or copy your prior version of the file into the configuration module before you can resolve the file.</p> <p>Double-clicking opens the file in the merge tool specified by <code>upgrader.merge.tool</code> in <code>upgrade.properties</code> to perform a three-way merge between your custom version and the updated Guidewire version. If you make changes, the tool prompts you to copy the merged file to the target configuration module.</p>

Merge Status	Guidewire Action	Your Action	Type of change made to file	Action taken by Configuration Upgrade Tool
EDIT_DELETE	delete	edit	file changed from base in customer configuration and does not exist in target configuration	<p>If you resolve the file, the Configuration Upgrade Tool takes no action.</p> <p>Double-clicking the file opens your customized file and the original base file in the merge tool specified by <code>upgrader.diff.tool</code> in <code>upgrade.properties</code>. When you close the merge tool, the Configuration Upgrade Tool prompts you to copy the file to the target configuration module. If you are sure you want your customized version, you can click Copy prior customized to move the file to the configuration module of the target version.</p> <p>The EDIT_DELETE flag appears on a file when your configuration has a customized version of the file but Guidewire has deleted the file from that location. There are two possible reasons for this deletion. One reason is that Guidewire removed the file from ClaimCenter. The second reason is that Guidewire has moved the file to a different folder.</p> <p>If Guidewire has completely removed the file, review the <i>ClaimCenter New and Changed Guide</i>, release notes, and the Upgrade Diffs report for descriptions of the change affecting the deleted file. Then determine if you want to continue moving your customization to the new or changed feature. If not, then the customization will be lost.</p> <p>For the second scenario, find where the file has been moved by searching the target version. Move your customized file to the same location in the working directory and make sure to match any case changes in the file-name. When you refresh the list of merge files, the file now appears under the CUSTOMER_EDIT filter. You can now proceed with the merge. If you do not move the file over, you can instead perform the merge manually by opening both files and incorporating the changes.</p>
NO_CHANGE	none	none	file not changed from base configuration in either customer or target configurations	<p>If you resolve the file, the Configuration Upgrade Tool takes no action. The file already exists in the target configuration.</p> <p>Double-clicking opens the file in the text editor specified by <code>upgrader.editor.tool</code> in <code>upgrade.properties</code>. If you make changes, the tool prompts you to copy the file to the target configuration module.</p>

Resolved Status Checkboxes

Beneath the change status filters are checkboxes to toggle the visibility of resolved and unresolved files. Use these checkboxes with the change status filters to specify which types of files you want visible in the center panel. For example, you could select **BOTH_EDIT** and **Unresolved** to see files edited in your configuration that have also been updated by Guidewire and are not yet resolved.

The purpose of the resolved status is to have a general idea of the progress you are making in the upgrade. The tool shows the resolved status of the current file (right panel) and the total number of resolved and unresolved files (left panel).

A resolved file is simply a file that you have marked resolved. It does not relate to whether file merging or accepting has occurred.

Bulk Action Buttons

The following buttons in The **Bulk Action** part of the left panel enable you to change the resolved status of a group of selected files:

- **Copy/Mark Resolved**
- **Mark Unresolved**

You can select either one or several files and directories before using these buttons. Use the CTRL key to select multiple files and directories. Selecting a directory selects all files within that directory. You can select all files that match the filters you set by selecting the top-level directory.

After you click **Copy/Mark Resolved**, the Configuration Upgrade Tool opens a dialog detailing the actions it is about to perform.

The tool copies files matching the **CUSTOMER_ADD** and **CUSTOMER_EDIT** filters to the target configuration module. If there is already a version of a file in the target configuration module, then the tool does not copy the file. A file would be there already if you edited the file and clicked **Yes** when the tool prompted you to copy the file to the configuration module.

The tool does not do any copying for files matching the **GW_ADD**, **GW_DELETE**, **GW_EDIT**, **NO_CHANGE**, or **Unmergeable** filters. Files matching **GW_ADD**, **GW_EDIT**, **NO_CHANGE**, or **Unmergeable** filters are already present in the target version. Files matching the **GW_DELETE** filter are not in ClaimCenter 6.0.8.

You can not bulk resolve multiple files that match the **BOTH_ADD**, **BOTH_EDIT**, or **EDIT_DELETE** filters. Files matching these filters require individual attention. Use the right panel of the Configuration Upgrade Tool to control merging, copying and resolving of these files.

Configuration File Tree

The center panel displays the configuration file tree. Files are color-coded to match filter colors. Files are shown one time, regardless of the number of configurations in which they exist. For information on which configurations a file exists in, select the file and view the right panel. The number of files in each directory that match the selected change status and resolved status filters is shown in parentheses.

File Details Panel

The right panel displays file details for the file you are currently examining, including:

- **Status:** the change status of the file. See “Change Status Filters” on page 203.
- **New base file:** The new version of this file supplied by Guidewire, typically in either the **cc** or **p1** module of ClaimCenter 6.0.8. If there is not a Guidewire version of this file, such as for **CUSTOMER_ADD**, **EDIT_DELETE** or **GW_DELETE** files, this field is blank.
- **Prior customized file:** The locally customized version of this file from the prior version. If there is not a customized version of this file, such as for **GW_ADD**, **GW_DELETE**, **GW_EDIT** or **NO_CHANGE** files, this field is blank.
- **Prior base file:** The base version of this file in the working directory. If there is not a Guidewire version of this file in the prior base version you are upgrading from, such as for **CUSTOMER_ADD** files, this field is blank.
- **New customized file:** The customized version of this file in the ClaimCenter 6.0.8 configuration directory.

The right panel fields are blank if you have multiple files selected.

File Details Panel Actions

The following buttons appear below the file details display in the right panel after you have selected a file:

- **View:** Opens the file in the editor specified in `upgrade.properties`. This button appears for files that are not customized and do not require merging, matching **GW_ADD**, **GW_EDIT**, or **GW_DELETE** filters. Only one of the **View**, **Edit** or **Merge** buttons displays, depending on the file change status.
- **Edit:** Opens the file in the editor specified in `upgrade.properties`. This button appears for custom files that do not require merging, matching the **CUSTOMER_ADD** or **EDIT_DELETE** filters.

- **Merge:** Opens the different versions of the file in the merge tool specified in `upgrade.properties`. This button appears for files that require merging, matching the **BOTH_ADD** or **BOTH_EDIT** filters.
- **Copy prior customized:** Copies the prior customized version of the file to the target configuration module. This button is enabled if there is a prior customized version of the file. So it is enabled for files matching **CUSTOMER_ADD**, **CUSTOMER_EDIT**, **BOTH_ADD**, **BOTH_EDIT**, or **EDIT_DELETE** filters.
- **Delete new customized:** Remove the customized version from the configuration module. This reverses the **Copy prior customized** button action. This button is disabled until you have copied a prior customized version of the file into the configuration module.
- **Resolved:** Check this box to label the file resolved. Use the **Resolved** checkbox in the right pane to change the status of a single file. Selecting the **Resolved** checkbox does not copy the file. Use the buttons above this checkbox to handle copying or merging of file versions. You must first unresolve a file before either using the **Delete new customized** action or reapplying changes or merges.

Accepting Files that Do Not Require Merging

The following filters show lists of files that normally do not require merging.

- **CUSTOMER_ADD**
- **CUSTOMER_EDIT**
- **GW_ADD**
- **GW_EDIT**
- **NO_CHANGE**
- **Unmergeable**

You can click the **Copy/Mark Resolved** button in the left panel to resolve groups of these files.

Merging and Accepting Files

Files matching the **BOTH_ADD** and **BOTH_EDIT** filters must be merged before being accepted. Your version must be reconciled with the Guidewire target or base version. In some cases, even if only a single version of the file exists, you might want to look at it before accepting it.

You can use the `pcf.xsd` file in `modules/cc/config/web/schema` of the target version to validate merged PCF files.

After you are satisfied with any changes, save the file. This saves the file in a temporary directory. When you close the editor or merge tool, the Configuration Upgrade Tool asks if you want to copy the file to the target configuration module. If you click **Yes**, the tool copies the file. If you click **No** (or **CTRL+N**), the tool cancels the popup without copying. The tool always moves files into the configuration module, except if a file is identical to the base or target version. In this case, the tool does not move the file.

Note: Do not edit a file version with **DO_NOT_EDIT** in its file name.

Configuration Merging Guidelines

First, work to generate the toolkit. The first milestone of an upgrade project is to generate a toolkit (java-api and soap-api) on the target release. To do this, you must:

- Complete the merge of the data model. This includes all files in the `/extensions` and `/fieldvalidators` folders.
- Resolve issues encountered while trying to generate the toolkit or start the QuickStart application server.

You can generate a toolkit even if you have errors in your enhancements, rules and PCF files.

Typical errors

- **Malformed XML:** The merge tool is not XML-aware. There might be occasions in which the file produced contains malformed XML. To check for well-formed XML, use free third-party tools such as Liquid XML, XML Marker, or Eclipse.
- **Duplicate typecodes:** As part of the merge process, you might have inadvertently merged in duplicate, matching typecodes.
- **Missing symmetric relationship on line-of-business-related typelists:** You might be missing a parent-child relationship with respect to the line-of-business-related typelist, as a result of merging.
- **References to GScript instead of Gosu,** such as in plugins/registry.

Once the toolkit has been generated, you can begin the work of upgrading integrations. See “Upgrading Integrations and Gosu from 4.0.x” on page 293.

Second, after you can successfully generate the Java and SOAP APIs, work on starting the server.

In addition to the typical issues above, the server might fail to start due to cyclical graph reference errors. See “Identifying Data Model Issues” on page 236.

You can generate a toolkit even if you have errors in your enhancements, rules and PCF files, although the error messages will print upon server startup.

Once the server can start on the target release, you can begin the database upgrade process.

Continue with the remainder of the configuration upgrade work, which includes evaluating existing PCF files and merging in desired changes.

Data Model Merging Guidelines

From a purely technical standpoint, not addressing the need to incorporate new features, the following are a few guidelines for merging the data model.

Merging Typelists: Overview

There is no automated process to merge typelists. This is a part of the merge process using the Configuration Upgrade Tool. In general, merge typelists before PCF files.

Merge in Guidewire-provided typecodes related to lines of business and retire unused typecodes that you merge in. If you do not include these typecodes, you will have errors in any enhancements, rules, or PCF files that reference the typecode. This also simplifies the process for future upgrades as there will be fewer added line of business typecodes to review.

Pay particular attention if any Guidewire-provided typecodes have the same typecode as a custom version. In this case, modify one of the typecodes so they are unique. Contact Guidewire Support for details.

The Configuration Upgrade Tool displays most typelists you have edited in the `CUSTOMER_EDIT` filter. If your edits are simply additional typecodes, accept your version.

Use Guidewire Studio to verify PCF files, enhancements, and rules to identify any issues with the files and rules that reference typelists.

IMPORTANT You might have edited a typelist with internal typecodes that you wanted to override. To change such typecodes, you must have first made a copy of the typelist. Such a file now appears in the `CUSTOMER_ADD` section, and must be merged. Before accepting the custom version, check that the change does not cause the application to have problems.

Merging Typelists: Simple Typelists

Merge in new typecodes from the target version, ClaimCenter 6.0.8. If you do not merge the new typecode, you will have errors in any enhancements, rules, or PCF files that reference the typecode. If you do not want to use a new typecode, retire the typecode by setting the `retired` attribute to `true`.

In the following example, the target version of ClaimCenter (shown on left) introduced the `debrisremoval` typecode.



To avoid errors in enhancements, rules, and PCF files that reference the `debrisremoval` typecode, merge the typecode. If you do not want to use the new typecode, merge it into the target file and retire it by setting the `retired` attribute to `true`, as shown below.

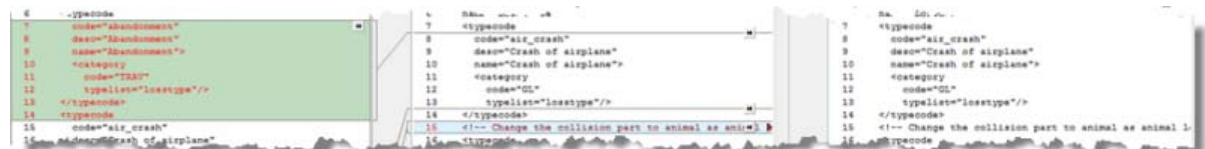


Merging Typelists: Complex Typelists

A typecode can reference typecode values from another typelist using the `<category>` subelement. If a new typecode references an existing typecode, do not merge the new typecode unless the referenced typecode is retired. Otherwise, you are defining a new relationship. If the referenced typecode is also new, merge in both typecodes. If you do not want to use a new typecode, set the `retired` attribute for the typecode to `true`. The following table summarizes how to handle merging new typecodes that reference other typecodes:

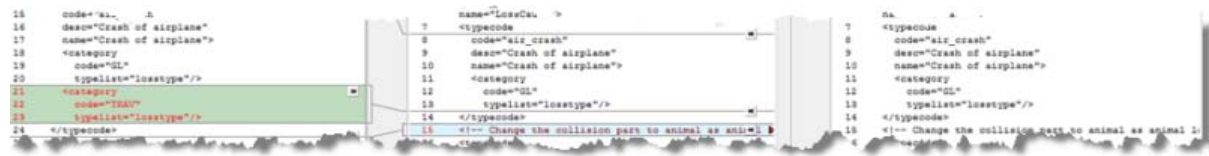
Referenced typecode status	Action
new: exists only in target version	Merge in the new typecode and merge in the referenced typecode in its typelist. If you do not want to use the new typecode, retire it by setting the <code>retired</code> attribute of the typecode to <code>true</code> .
active: exists in base or custom version and is not retired	Do not merge the new typecode.
retired: exists in base or custom version and is retired	Merge in the new typecode. If you do not want to use the new typecode, retire it by setting the <code>retired</code> attribute of the typecode to <code>true</code> .

In the following example, the typecode `abandonment` did not exist previously, either in the old default configuration or in the custom configuration. If you do not want the new typecode, Guidewire recommends that you merge and retire the typecode and all its children. This assumes that you are following other recommendations such that the `TRAV loss` type referenced is also merged in, and retired if applicable. Because this typecode is retired, there are no harmful consequences of this action. If you choose to unretire the typecode in the future, you will have the desired, default behavior in place.



In the next example, `air_crash` is a typecode that is already existing and active in the original base version. The latest version introduces a new `<category>` `loss` subelement of `TRAV` to the `air_crash` typecode. If `TRAV` is

already an active typecode, do not merge in the new <category> subelement. If you do, you will create a line of business relationship in the data model which did not previously exist. Merging in category subelements that reference active typecodes might produce unwanted consequences.



Note: The LineCategory typelist tends to be very large. Guidewire recommends that you select the **Copy Prior Customized** option and not merge the LineCategory typelist. If you are keeping your existing line of business configuration, your customized LineCategory typelist is sufficient for the upgrade.

Merging Entities

Guidewire recommends that you merge in newly defined indexes from the target configuration.

For example, in the Claim.etx file, merge in the following changes to your customized configuration:

```
<index
  desc="Covering index for helping to speed up Claim Search by Insured's last name, when including
        lossdate as one of the criteria"
  expectedtobecovering="true"
  name="claimu6"
  trackUsage="true"
  unique="true">
  <indexcol
    keyposition="1"
    name="InsuredDenormID"/>
  <indexcol
    keyposition="2"
    name="Retired"/>
  <indexcol
    keyposition="3"
    name="LossDate"/>
  <indexcol
    keyposition="4"
    name="ID"/>
</index>
```

Merging Other Files

In some cases, the differences between files cannot be merged effectively using a comparison tool. In particular, config.xml, logging.properties, and scheduler-config.xml often have many changes between major versions. Consider adding your custom changes to the new Guidewire-provided version instead of merging from prior versions if the presentation of these files in the merge tool is too daunting.

IMPORTANT ClaimCenter 6.0.8 includes a StrictDataTypes parameter in config.xml. This parameter controls validation of data by ClaimCenter. The default value for StrictDataTypes is true. Guidewire recommends that you set StrictDataTypes to false to perform the upgrade. Setting StrictDataTypes to false preserves the existing data validation behavior. For more information about the StrictDataTypes parameter, see “StrictDataTypes” on page 55 in the *Configuration Guide*.

Upgrading Rules

The Configuration Upgrade Tool upgrades rules to Gosu classes instead of the former XML format. You cannot perform a merge on rules because of this format change. Instead, after merging the rest of the configuration, use Studio to compare the ClaimCenter 6.0.8 default rules with your rules and update as necessary.

The automated upgrade process moves your custom rulesets into the modules\configuration\ subfolder structure. This has the effect of adding at least 21 characters to rule file names. If you have long rule names, you might

reach the Windows file name length limit of 255 characters because of the addition of `modules\configuration` letters in your path. To avoid this, you can use a combination of:

- Shortening rule names for those rules that have the potential of reaching the limit.
- Keeping your workspace closer to the root folder (`C:\`).
- Applying a combination of these approaches. Additionally, to make this and future upgrades more efficient, consider removing unused, inactive rules. The automated upgrade process will copy a set of default rules for reference. You can use this to review preferred coding styles or to copy rules around new features. You can delete this reference folder after the upgrade work is complete.

See “Upgrading Rules to ClaimCenter 6.0.8” on page 222.

About Display Properties

You do not need to manually merge `display.properties`. ClaimCenter collects display properties from both the `cc` and `configuration` modules. This is different than the way ClaimCenter handles most other files. For most files, if a version exists in the `configuration` module, ClaimCenter uses that version instead of the version in the `cc` module. For `display.properties`, ClaimCenter creates a union of values from both files.

During the upgrade, any custom display properties are moved to the `configuration` module. Any display properties that have been added by Guidewire are in the `cc` module. Although `display.properties` appears in the Configuration Upgrade Tool interface, you do not need to merge this file. ClaimCenter essentially performs the merge for you each time it starts.

If you have added locales, you can export a full list of display keys and typelists from the default ClaimCenter 6.0.8 locale to any locale you have defined. This list includes a section for display keys and typelists that do not yet have values defined for your locale. You can use this list to determine which display keys and typelists require localized values. You can then specify those values and import the list.

See “Translating New Display Properties and Typecodes” on page 224.

Merging Classes (Libraries)

Merging libraries is the same as merging other configuration files. The configuration upgrade tool converts all libraries to classes and places them in the `/config/resources/classes/libraries` folder.

Use Studio to find and fix validation problems in Gosu classes and enhancements. This process is described in a later step.

Method Removed from Typekeys

In ClaimCenter 5.0.4, Guidewire removed the `valueOf` method that was present on all typekeys. Replace calls to the `valueOf` method with calls to the `getByCode` method.

Merging PCF Files

This topic provides some guidance for merging PCF file changes for specific features. Review the PCF file changes described in the automated steps for changes that apply to all PCF files.

General PCF Changes

You can no longer create a PCF page using a Search Panel widget as the top-level element. You can, however, still use this widget as a lower-level element. None of the PCF pages provided with previous versions of ClaimCenter used a Search Panel widget as the top-level element. However, you might have created such a PCF

page in your custom configuration. If you have, modify the PCF page so that the Search Panel widget is not the top-level element.

Address Consolidation

ClaimCenter had multiple views to display addresses for different formats and entities. For example, the view for `Claim.LossLocation` was different from `Exposure.TempLocation`. Both of these are addresses, but the views used were different.

Different address views posed a challenge for maintenance and localization. For example, if you did not want to display **Address Line 2**, you needed to configure multiple views. Similarly, if you needed to add an international address format for another country, that too required you to configure multiple files.

To resolve this issue, ClaimCenter 6.0 includes a single address view that can have modes for each country code. This provides one view that you can customize for the examples stated above.

The new `AddressOwner` interface specifies which entity owns a particular address, so that the address can be saved properly. For example, if the address is a loss location, it must be saved on `Claim`. If it is a temporary location for loss of use, then it must be saved on `Exposure`. All entities with addresses extend this interface. Custom extensions are supported.

AddressOwner and CCAddressOwner

The `AddressOwner` is the interface to create a helper object that is passed to the `AddressInputSet`. The helper object provides a way to set and get a single address on the enclosing entity. It also provides methods to allow fields to be required, visible, and so forth.

The `CCAddressOwner` interface extends the `AddressOwner` interface to add ClaimCenter specific fields. A `CCAddressOwner` object is passed as the argument to the generic `AddressInputSet`. The generic `AddressInputSet` calls methods and properties on `CCAddressOwner` to determine which fields to show, which fields are required, how to get and set the Address object, and so forth. This allows a single modal `AddressInputSet` to be reused in many different situations, making it easier to add new countries to the system.

Overall, there are different levels at which you can configure address fields:

- Simple global changes can be made by altering the constant sets of values in `CCAddressOwnerFieldId`.
- Country-specific changes can be made by adding a country-specific `CountryAddressFields` object. Refer to the `CountryAddressFields` class for details.
- Changes specific to a particular use of `AddressInputSet` can be made by adding or altering the particular `CCAddressOwner` passed into that input set. This owner object can override any country-specific or global defaults.

New Address User Interface

The new user interface is intended to be consistent across ClaimCenter with a few exceptions as noted.

In view mode, ClaimCenter always displays the address on two lines. This was done by adding a new line to the Address entity name. The description field is now available for all addresses.

In edit mode, the address section looks similar to earlier versions of ClaimCenter.

The address user interface always displays a range selector to allow selection from an appropriate address array. If the address is not editable, the fields are grayed out. The range selector can display **New...** to allow creation of a new address.

Address PCF Configuration

The shared address views are located under `/cc/config/web/pcf/claim/shared/address`.

Each `AddressInputSet.XX.pcf` relates to a locale and is referenced using an `InputSetRef` widget. For instance, from `LossDetailsDV.Auto.pcf`:

```
<InputSetRef
  def="AddressInputSet(Claim.AddressOwner)"
  mode="Claim.AddressOwner.InputSetMode" />
```

In the example, the `Claim.AddressOwner` parameter passed to `AddressInputSet` is a `CCAddressOwner` object. The mode is the switch to select the correct view mode. If a country selected from the drop-down does not exist as a mode, ClaimCenter uses `AddressInputSet.default.pcf`.

Gosu Configuration for a New Address Relationship

You might need to delegate an entity as an `AddressOwner` when you add an `Address` foreign key to:

- an existing entity
- a new entity

For both cases:

1. First create a new Gosu class that implements the `AddressOwner` or `CCAddressOwner` interface. Studio prompts you to implement the methods.
2. Press ALT + ENTER to implement the methods. See `VehicleIncidentAddressOwner.gs` and `ClaimRelatedAddressOwner.gs` for an implementation example.
3. Create a new Gosu enhancement to enhance the new or existing entity with `get` and `set` methods. See `GWTripAccommodationAddressOwnerEnhancement.gsx` for sample implementation details.

Address Files Removed

The following redundant files were removed as a part of the address consolidation:

- `LossDetailsAddressDetailInputSet.default.pcf`
- `LossDetailsAddressDetailInputSet.CA.pcf`
- `LocationDetailInputSet.CA.pcf`
- `LocationDetailInputSet.US.pcf`
- `LocationDetailInputSet.default.pcf`
- `LossOfUseTempAddressInputSet.CA.pcf`
- `LossOfUseTempAddressInputSet.US.pcf`
- `LossOfUseTempAddressInputSet.default.pcf`
- `AddressBookContactAddressDetailInputSet.CA.pcf`
- `AddressBookContactAddressDetailInputSet.US.pcf`
- `AddressBookContactAddressDetailInputSet.default.pcf`
- `UserAddressDetailInputSet.CA.pcf`
- `UserAddressDetailInputSet.US.pcf`
- `UserAddressDetailInputSet.default.pcf`
- `ContactAddressDetailInputSet.CA.pcf`
- `ContactAddressDetailInputSet.US.pcf`
- `ContactAddressDetailInputSet.default.pcf`
- `FixedPropertyIncidentDV.CA.pcf`
- `FixedPropertyIncidentDV.default.pcf`
- `AddressDetailDV.CA.pcf`
- `AddressDetailDV.US.pcf`
- `AddressDetailDV.default.pcf`

Multiple inline addresses screens have been updated to use the `AddressInputSet` to further ease address configuration across the system.

ContactCenter Address Input

ContactCenter 6.0.8 no longer includes the following address input PCF files: `AddressDetailInputDVSet.pcf` and `AddressAutofillableInputSet.pcf`. If you have custom PCF files that reference these files, update the references to instead use `AddressDetailInputSet.pcf` or `AddressAutofillableDVInputSet.pcf`, respectively. If you customized `AddressDetailInputDVSet.pcf` or `AddressAutofillableInputSet.pcf`, review `AddressDetailInputSet.pcf` and `AddressAutofillableDVInputSet.pcf`, respectively and apply any required customizations.

Homeowners FNOL

This section lists the files that Guidewire has added, changed or removed as part of the change to the homeowners FNOL. You might choose not to use the new homeowners FNOL because you already have customized FNOL files for a homeowners line of business.

All file names are given from the base of `/cc/config/web/pcf`.

Files Added

All the `/claim/snapshot/600` pages are new. These files include changes to display homeowner data. These snapshot PCF files are not included in the list below.

Other files added for homeowners FNOL include:

- `/claim/FNOL/FNOLContactInputSet.pcf`
- `/claim/FNOL/FNOLInjuryIncidentPopup.pcf`
- `/claim/FNOL/FNOLWizard_BasicInfoPolicyPanelSet.Homeowners.pcf`
- `/claim/FNOL/FNOLWizard_BasicInfoRightPanelSet.Pr.pcf`
- `/claim/FNOL/FNOLWizard_NewLossDetailsPanelSet.homeowners.pcf`
- `/claim/FNOL/FNOLWizard_NewLossDetailsScreen.PR.pcf`
- `/claim/FNOL/FNOLWizard_ServicesPolicyPanelSet.Homeowners.pcf`
- `/claim/FNOL/FNOLWizard_ServicesScreen.Auto.pcf`
- `/claim/FNOL/FNOLWizard_ServicesScreen.Pr.pcf`
- `/claim/FNOL/FixedPropertyIncidentDebrisRemovalInputSet.pcf`
- `/claim/FNOL/FixedPropertyIncidentEMSInputSet.pcf`
- `/claim/FNOL/NewClaimContentsDamageDV.pcf`
- `/claim/FNOL/NewClaimExposureDV.Content.pcf`
- `/claim/FNOL/NewClaimExposureDV.LivingExpenses.pcf`
- `/claim/FNOL/NewClaimExposureDV.OtherStructure.pcf`
- `/claim/FNOL/NewClaimLivingExpensesDV.pcf`
- `/claim/FNOL/NewClaimOtherStructureDamageDV.pcf`
- `/claim/exposures/ContentsDamageDV.pcf`
- `/claim/exposures/ExposureDetailDV.Content.pcf`
- `/claim/exposures/ExposureDetailDV.LivingExpenses.pcf`
- `/claim/exposures/ExposureDetailDV.OtherStructure.pcf`
- `/claim/exposures/ExposureDetailPopup.pcf`
- `/claim/exposures/LivingExpensesDV.pcf`
- `/claim/exposures/OtherStructureDamageDV.pcf`
- `/claim/lossdetails/EditableRoomsLV.pcf`
- `/claim/lossdetails/FireDamageInfoInputSet.pcf`
- `/claim/lossdetails/FireDamageQuestionsPanelSet.pcf`
- `/claim/lossdetails/LossDetailsPanelSet.Pr.pcf`
- `/claim/lossdetails/LossDetailsPanelSet.pcf`
- `/claim/lossdetails/LossDetailsPrPanelSet.homeowners.pcf`
- `/claim/lossdetails/LossDetailsPrPanelSet.pcf`
- `/claim/lossdetails/RelatedExposuresLV.pcf`
- `/claim/lossdetails/WaterDamageQuestionsPanelSet.pcf`

- /claim/lossdetails/print/DwellingIncidentPrint.pcf
- /claim/lossdetails/print/InjuryIncidentDetailPrint.pcf
- /claim/lossdetails/print/InjuryIncidentsPrint.pcf
- /claim/lossdetails/print/LivingExpensesIncidentPrint.pcf
- /claim/lossdetails/print/OtherStructureIncidentPrint.pcf
- /claim/lossdetails/print/PropertyContentsIncidentPrint.pcf
- /claim/newexposure/NewExposureDV.Content.pcf
- /claim/newexposure/NewExposureDV.LivingExpenses.pcf
- /claim/newexposure/NewExposureDV.OtherStructure.pcf
- /claim/planofaction/ClaimEvaluationDetailsDV.Trav.pcf
- /claim/shared/Property/PropertyAddressInputSet.CA.pcf
- /claim/shared/Property/PropertyAddressInputSet.US.pcf
- /claim/shared/Property/PropertyAddressInputSet.default.pcf

Files Edited

- /admin/reinstthreshold/ReinsuranceThresholdLV.pcf
- /claim/FNOL/FNOLContactPopup.pcf
- /claim/FNOL/FNOLWizard.pcf
- /claim/FNOL/FNOLWizardFindPolicyPanelSet.Search.pcf
- /claim/FNOL/FNOLWizard_BasicInfoRightPanelSet.Auto.pcf
- /claim/FNOL/NewClaimExposureDV.Propertydamage.pcf
- /claim/FNOL/NewClaimLossDetailsDV.Auto.pcf
- /claim/FNOL/NewClaimLossDetailsDV.GL.pcf
- /claim/FNOL/NewClaimLossDetailsDV.Pr.pcf
- /claim/FNOL/NewClaimLossDetailsDV.Wc.pcf
- /claim/lossdetails/ClaimLossDetails.pcf
- /claim/lossdetails/DwellingIncidentPanelSet.pcf
- /claim/lossdetails/LossDetailsDV.GL.pcf
- /claim/lossdetails/LossDetailsDV.Pr.pcf
- /claim/lossdetails/LossDetailsPanelSet.Homeowners.pcf
- /claim/lossdetails/OtherStructureIncidentPanelSet.pcf
- /claim/lossdetails/PrDV.pcf
- /claim/lossdetails/PropertyAttributeInputSet.pcf
- /claim/lossdetails/PropertyContentsIncidentPanelSet.pcf
- /claim/medicalcasemgmt/ClaimWCmedCaseMgmt.pcf
- /claim/summary/indicator/ClaimIndicatorInputSet.FlagClaimIndicator.pcf
- /shared/printing/ClaimPrintout.pcf

Files Deleted

- /claim/FNOL/FNOLWizard_ServicesScreen.pcf
- /claim/lossdetails/FixedPropertyIncidentDV.CA.pcf
- /claim/lossdetails/FixedPropertyIncidentDV.default.pcf

Fixed Property Incident Detail View

In FixedPropertyIncidentDV.pcf the LienHoldersLV list view is set to be visible only if the PolicyType is not homeowners. To see the data for homeowners policies remove the visible attribute from the ListViewInput parameter.

```
<ListViewInput
def="EditablePropertyLienholdersLV(FixedPropertyIncident.Property, Claim)"
editable="Claim.canEditProperty(FixedPropertyIncident.Property)"
label="displaykey.Web.FixedPropertyIncident.Property.Lienholders"
validationExpression="FixedPropertyIncident.Property != null
FixedPropertyIncident.Property.validateLienholders() : null"
visible="FixedPropertyIncident.Claim.Policy.PolicyType != "homeowners"">
```

Incident Type-specific Behavior

ClaimCenter adds new exposure and incident types and maps these exposure and incident types to existing coverage subtypes. See “Remapping Incidents” on page 152. You can modify this mapping by changing the mapping of your coverage subtypes to exposure types.

The updated line of business configuration will result in incident subtypes changing. Be prepared for any type-specific behavior of the new incident type. For the new Dwelling incident type, one such behavior is that the loss location of the claim is used as the location for the incident. Any location explicitly set on an incident that becomes a Dwelling incident will not be visible in ClaimCenter.

By default, the ClaimCenter upgrade remaps Fixed Property Incidents with the following coverage subtypes to Dwelling incidents:

- Dwelling - PropertyDamage
- Earthquake - Dwelling - PropertyDamage
- Flood - Dwelling - PropertyDamage
- Mold - Dwelling - PropertyDamage

PCF File Changes Required if not Remapping Property Exposures

ClaimCenter remaps Property exposures mapped to the Fixed Property Incident type to Dwelling or Other Structure incident types, depending on the coverage subtype. See “Remapping Incidents” on page 152. You can modify this mapping before you run the database upgrade. If you change this mapping so that these Property exposures stay mapped to the Fixed Property Incident type, update PCF files that reference `Exposure.DwellingIncident` or `Exposure.OtherStructureIncident` to instead reference `Exposure.FixedPropertyIncident`.

The following PCF files reference `Exposure.DwellingIncident`:

- `config/web/pcf/claim/exposures/PropertyIncidentInputSet.Dwelling`
- `config/web/pcf/claim/FNOL/NewClaimIncidentInputSet.Dwelling`
- `config/web/pcf/claim/snapshot/600/ClaimSnapshotExposure600DV.Dwelling`

The following PCF files reference `Exposure.OtherStructureIncident`:

- `config/web/pcf/claim/exposures/OtherStructureDamageDV`
- `config/web/pcf/claim/exposures/NewClaimOtherStructureDamageDV`
- `config/web/pcf/claim/snapshot/600/ClaimSnapshotExposure600DV.OtherStructure`

Holidays

ClaimCenter 6.0 adds `ZoneType` and `Country` to `HolidayZone` entities. These fields help to more precisely define a `HolidayZone`. As businesses expand into more territories, this becomes necessary to retain unique `HolidayZones`. During holiday configuration, ClaimCenter 6.0 enables you to specify the zone type and country to which the holiday apply. You cannot use the holiday configuration PCF files from a prior version. Accept the new Guidewire versions of files in `config/web/pcf/admin/holidays`. If you want to make changes to any of these files, you can customize the new versions.

Regions

ClaimCenter 6.0 removes the deprecated `Region.RegionType` field. If you have customized region configuration PCF files that used this field, you must remove it.

ClaimCenter 5.0 PCF files did not include `Region.RegionType` by default.

ClaimCenter 4.0 included `Region.RegionType` by default in the following PCF files:

- `config/web/pcf/admin/regions/RegionDetailDV`
- `config/web/pcf/admin/regions/RegionsLV`

Catastrophes

ClaimCenter 6.0 changed from using `CatastropheState` entities to `CatastropheZone` entities. However, you can preserve `CatastropheState` data by creating an extension on `Catastrophe` prior to upgrading the database.

Guidewire has updated PCF files that reference `CatastropheState` and has removed the `config/web/pcf/admin/catastrophes/CatastropheStateLV` file. If you keep the `CatastropheState` entity, look for PCF files that have changed from referencing `CatastropheState` to `CatastropheZone` and include the `CatastropheState` references in the PCF files. In particular, Guidewire has removed `CatastropheState` from `config/web/pcf/admin/catastrophes/AdminCatastropheDV`.

You might have decided to switch from `CatastropheState` entities to `CatastropheZone` entities and decided to preserve `CatastropheState` comments. If you created the `Comments` column extension on `CatastropheZone`, you might want to add `CatastropheZone.Comments` to `catastrophe` PCF pages.

Deductible Fields

ClaimCenter 6.0 adds a new `Deductible` entity. The database upgrade removes the following deductible fields from `Claim`, `Transaction` and `Payment` if they are empty.

- `Claim.DeductibleStatus`
- `Transaction.DeductiblePaid`
- `Payment.DeductibleAmount`
- `Payment.DeductibleSubtracted`
- `Payment.NetAmount`
- `Payment.OriginalAmount`

However, if you have data in these fields that you want to keep, you can create extensions to preserve these fields in the database.

Guidewire has updated PCF files that reference these deductible fields. If you preserve the fields, keep the references to the deductible fields when merging PCF files.

Prior versions of ClaimCenter included `Claim.DeductibleStatus` on the following PCF files by default:

- `config/web/pcf/claim/lossdetails/LossDetailsDV.Auto`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.GI`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Pr`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Wc`

These references have been removed in ClaimCenter 6.0. None of the other deductible fields listed were referenced on ClaimCenter PCF files by default.

Claim snapshot PCF files show a claim as it was originally created. Claim snapshots are stored as XML rather than in the `cc_claim` table, so they are not affected by later changes to the `Claim` entity. So some claim snapshot PCF files include the `DeductibleStatus` because it was previously a property on `Claim`. These files include:

- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Auto`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.GI`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Pr`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Wc`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Auto`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.GI`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Pr`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Wc`

For more information on claim snapshots, see “How ClaimCenter Renders Claim Snapshots” on page 629 in the *Configuration Guide*.

Reinsurance Status

Guidewire has updated PCF files to remove references to `Claim.ReinsuranceStatus`. If you keep this field by creating an extension, look for PCF files that have `Claim.ReinsuranceStatus` references removed and add the reference.

Prior versions of ClaimCenter included `Claim.ReinsuranceStatus` in the following files by default:

- `config/web/pcf/claim/lossdetails/LossDetailsDV.Auto`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Gl`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Pr`
- `config/web/pcf/claim/lossdetails/LossDetailsDV.Wc`

These references have been removed in ClaimCenter 6.0.

Claim snapshot PCF files show a claim as it was originally created. Claim snapshots are stored as XML rather than in the `cc_claim` table, so they are not affected by later changes to the `Claim` entity. So some claim snapshot PCF files include the `ReinsuranceStatus` because it was previously a property on `Claim`. These files include:

- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Auto`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Gl`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Pr`
- `config/web/pcf/claim/snapshot/300/ClaimSnapshotGeneral300DV.Wc`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Auto`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Gl`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Pr`
- `config/web/pcf/claim/snapshot/310/ClaimSnapshotGeneral310DV.Wc`

For more information on claim snapshots, see “How ClaimCenter Renders Claim Snapshots” on page 629 in the *Configuration Guide*.

New Claim Wizard Changes Since 4.0

This section lists the files that Guidewire has added, changed or removed as part of changes to the new claim wizard since ClaimCenter 4.0.

In ClaimCenter 4.0, the new claim wizard files were stored in `/claim/newclaimwizard`. In ClaimCenter 5.0 and 6.0, these files are now in `/claim/FNOL`. Therefore, in an upgrade from ClaimCenter 4.0, the Configuration Upgrade Tool shows all files in `/claim/newclaimwizard` as deleted and all files in `/claim/FNOL` as new.

All file names are given from the base of `/app-cc/cc/config/web/pcf`.

Files Added

- `ClaimSummaryPopup.pcf`
- `FNOLContactInputSet.pcf`
- `FNOLContactPopup.pcf`
- `FNOLInjuryIncidentPopup.pcf`
- `FNOLMenuActions.pcf`
- `FNOLVehicleIncidentPopup.pcf`
- `FNOLWizard.pcf`
- `FNOLWizard_AssignSaveScreen.pcf`
- `FNOLWizard_AutoFirstAndFinalScreen.pcf`
- `FNOLWizard_BasicInfoPolicyPanelSet.Homeowners.pcf`
- `FNOLWizard_BasicInfoRightPanelSet.Auto.pcf`
- `FNOLWizard_BasicInfoRightPanelSet.Pr.pcf`
- `FNOLWizard_BasicInfoRightPanelSet.Trav.pcf`
- `FNOLWizard_BasicInfoScreen.default.pcf`
- `FNOLWizard_BasicInfoScreen.WC.pcf`

- FNOLWizard_DocumentsScreen.pcf
- FNOLWizard_EditNotePopup.pcf
- FNOLWizard_EditNoteScreen.pcf
- FNOLWizard_FindPolicyScreen.pcf
- FNOLWizard_NewLossDetailsPanelSet.homeowners.pcf
- FNOLWizard_NewLossDetailsScreen.default.pcf
- FNOLWizard_NewLossDetailsScreen.PR.pcf
- FNOLWizard_NewLossDetailsScreen.Trav.pcf
- FNOLWizard_NewLossDetailsScreen.WC.pcf
- FNOLWizard_NewNoteWorksheet.pcf
- FNOLWizard_NewQuickClaimScreen.pcf
- FNOLWizard_NotesScreen.pcf
- FNOLWizard_PickPolicyRiskUnitsScreen.pcf
- FNOLWizard_PolicySearchInputSet.CA.pcf
- FNOLWizard_PolicySearchInputSet.default.pcf
- FNOLWizard_SelectPolicyRiskUnitsPopup.pcf
- FNOLWizard_ServicesPolicyPanelSet.Homeowners.pcf
- FNOLWizard_ServicesScreen.Auto.pcf
- FNOLWizard_ServicesScreen.Pr.pcf
- FNOLWizardAutoFirstAndFinalPanelSet.INSURED.pcf
- FNOLWizardAutoFirstAndFinalPanelSet.VENDOR.pcf
- FNOLWizardCheckDV.notready.pcf
- FNOLWizardCheckDV.ready.pcf
- FNOLWizardFindPolicyPanelSet.Create.pcf
- FNOLWizardFindPolicyPanelSet.Search.pcf
- InjuryIncidentInputSet.pcf
- NewClaimBaggageDamageDV.pcf
- NewClaimContentsDamageDV.pcf
- NewClaimExposureDV.Baggage.pcf
- NewClaimExposureDV.Content.pcf
- NewClaimExposureDV.LivingExpenses.pcf
- NewClaimExposureDV.OtherStructure.pcf
- NewClaimExposureDV.TripCancellationDelay.pcf
- NewClaimIncidentInputSet.Dwelling.pcf
- NewClaimIncidentInputSet.Propertydamage.pcf
- NewClaimLivingExpensesDV.pcf
- NewClaimLocationsLV.Trav.pcf
- NewClaimOtherStructureDamageDV.pcf
- NewClaimPolicyGeneralPanelSet.Auto.pcf
- NewClaimPolicyGeneralPanelSet.GL.pcf
- NewClaimPolicyGeneralPanelSet.Pr.pcf
- NewClaimPolicyGeneralPanelSet.Trav.pcf
- NewClaimPolicyGeneralPanelSet.Wc.pcf
- NewClaimTripCancellationDV.pcf
- QuickClaimDV.QuickClaimAuto.pcf
- QuickClaimDV.QuickClaimProperty.pcf
- QuickClaimDV.TravelBaggageOnly.pcf
- QuickClaimDV.TravelCancelOnly.pcf
- TotalLossCalculatorPopup.pcf
- VehicleIncidentAppraisalDV.pcf
- VehicleIncidentAutoBodyDV.pcf
- VehicleIncidentPanelSet.pcf
- VehicleIncidentRentalDV.pcf
- VehicleIncidentTowingDV.pcf

Files Edited

- DuplicateClaimSearchResultsLV.pcf
- NewClaimDocumentsLV.pcf
- NewClaimDuplicatesWorksheet.pcf
- NewClaimExposureDV.Bodilyinjurydamage.pcf
- NewClaimExposureDV.Employerliability.pcf
- NewClaimExposureDV.Generaldamage.pcf
- NewClaimExposureDV.Lossofusedamage.pcf
- NewClaimExposureDV.Medpay.pcf
- NewClaimExposureDV.Pipdamages.pcf
- NewClaimExposureDV.Propertydamage.pcf
- NewClaimExposureDV.Wcinjurydamage.pcf
- NewClaimExposuresLV.pcf
- NewClaimLocationsLV.Auto.pcf
- NewClaimLocationsLV.GL.pcf
- NewClaimLocationsLV.Pr.pcf
- NewClaimLocationsLV.Wc.pcf
- NewClaimLossDetailsDV.Auto.pcf
- NewClaimLossDetailsDV.GL.pcf
- NewClaimLossDetailsDV.Pr.pcf
- NewClaimLossDetailsDV.Wc.pcf
- NewClaimLostWagesBenefitsDV.pcf
- NewClaimLostWagesSummaryDV.pcf
- NewClaimNewExposureMenuItemSet.pcf
- NewClaimPeopleDV.pcf
- NewClaimPeopleInvolvedDetailedLV.pcf
- NewClaimPIPSummaryDV.pcf
- NewClaimPIPVocBenefitsDV.pcf
- NewClaimPropertyDamageDV.pcf
- NewClaimReplacementServicesDV.pcf
- NewClaimSummaryPeopleInvolvedLV.pcf
- NewClaimThirdPartyPropertyDamageDV.pcf
- NewClaimTowOnlyDV.pcf
- NewClaimVehicleDamageDV.pcf
- NewClaimVehiclesLV.pcf
- NewClaimWizard_DocumentsScreen.pcf
- NewClaimWizard_EndorsementDetailPopup.pcf
- NewClaimWizard_ExposurePagePopup.pcf
- NewClaimWizard_NewEndorsementPopup.pcf
- NewClaimWizard_NewExposurePopup.pcf
- NewClaimWizard_NewPolicyLocationPopup.pcf
- NewClaimWizard_NewPolicyVehiclePopup.pcf
- NewClaimWizard_NewStatCodePopup.pcf
- NewClaimWizard_PartyInvolvedPopup.pcf
- NewClaimWizard_PolicyDetailsScreen.pcf
- NewClaimWizard_PolicyGeneralScreen.pcf
- NewClaimWizard_PolicyLocationDetailPopup.pcf
- NewClaimWizard_PolicyVehicleDetailPopup.pcf
- NewClaimWizard_QuickClaimScreen.pcf
- NewClaimWizard_StatCodeDetailPopup.pcf
- PriorClaimsLV.pcf
- QuickClaimDV.AutoFirstAndFinal.pcf
- QuickClaimDV.GL.pcf

Files Deleted

- NewClaimExposureDV.Theft.pcf
- NewClaimPolicyGeneralDV.Auto.pcf
- NewClaimPolicyGeneralDV.GL.pcf
- NewClaimPolicyGeneralDV.Pr.pcf
- NewClaimPolicyGeneralDV.Wc.pcf
- NewClaimPolicySelectPolicyPopup.pcf
- NewClaimSelectTypeDV.pcf
- NewClaimTheftOnlyDV.pcf
- NewClaimWizard.pcf
- NewClaimWizard_NewNoteWorksheet.pcf
- NewClaimWizard_SelectTypeScreen.pcf
- QuickClaimDV.Auto.pcf
- QuickClaimDV.Pr.pcf
- QuickClaimDV.WC.pcf
- TypeofPropertyLV.pcf

Modifying New Label Wrapping Behavior

In ClaimCenter 6.0.8, input labels on PCF files can wrap more aggressively than they used to. Sometimes each word of a label can wrap to a new line. One important benefit of this new layout is that screens are less likely to scroll horizontally. Previously, content that did not fit on screen overflowed and caused the screen to scroll. Pages now attempt to fit all detail view content onto the screen, principally by wrapping labels in detail views, since the value field often cannot wrap, especially in edit mode.

While the new layout helps with horizontal scrolling, the wrapping can be extreme. On some screens, labels may wrap down to individual words. You can modify the new wrapping behavior if you do not like it. Consult Guidewire Software Knowledge Base article KB 436, “User Interface Spacing and Label Wrapping Post Upgrade” for instructions.

Merging Document, Email and Note Templates

Guidewire has updated templates in `config/resources/` for document, email, and note production to support localized versions. Guidewire has deprecated a number of template methods. These methods continue to work in ClaimCenter 6.0.8, but will be removed in a future release.

Merging Contact Role Constraints

During the automated configuration upgrade from ClaimCenter 4.0 to 5.0, the tool extracts information from the customer configuration `tl_cc_claim.xml`, `RoleConstraints.xml` and `ContactRole.xml` files and generates `entityroleconstraints-config.xml`. Merge this version with the default version provided in the target configuration. The tool removes the `RoleConstraints.xml` file and edits `ContactRole.xml` to remove all Category tags from the codes within the file.

See “Upgrading Contact Role Constraints” on page 187.

Merging ContactCenter Integration Files

During the configuration upgrade from ClaimCenter 4.0 to 5.0, you updated certain files used for integration with ContactCenter. See “Updating ContactCenter Integration Files” on page 190. These files are `cc-to-ab-data-mapping.xml` and `ab-to-cc-data-mapping.xml`. After you run the automated upgrade from 5.0 to 6.0, merge your updated 5.0 versions of these files with the 6.0 versions.

Upgrading Rules to ClaimCenter 6.0.8

The Configuration Upgrade Tool does not upgrade rules. The tool classifies rules in the unmergeable filter. Within the target directory, Guidewire-provided default rules are located in `modules/cc/config/rules` and `modules/pl/config/rules`. The Configuration Upgrade Tool moves your custom rules to `modules/configuration/config/rules`.

Guidewire also copies the default rules for the current release into a ClaimCenter 6.0.8 folder within `modules/configuration/config/rules/rules`. Update your rules in Studio. You can use the default rules in the ClaimCenter 6.0.8 folder as a comparison. Compare your custom rules to the new default 6.0.8 versions and update your rules as needed.

You might find it useful to do a bulk comparison of default rules from the base release against the 6.0.8 versions to determine what types of changes Guidewire has made.

To compare rules between versions using the Rule Repository Report

1. If you want to compare default rules only, temporarily remove custom rules from your starting version by moving the `modules/configuration/config/rules` directory to a location outside the ClaimCenter directory.

If you want to compare your custom rules against the ClaimCenter 6.0.8 rules, do not move the `modules/configuration/config/rules` directory from your starting version. However, do remove the `ClaimCenter<base version>` directory from `modules/configuration/config/rules/rules` of the starting version if this directory exists.
2. Open a command window.
3. Navigate to the `bin` directory of your starting version.
4. Enter the following command:

```
gwcc regen-rulereport
```


This command creates a rule repository report XML file in `build/cc/rules`.
5. Append the starting version number to the XML file name.
6. Restore moved directories to the starting version.
7. Temporarily copy `modules/configuration/config/rules` from the target version to a location outside the ClaimCenter directory. Removing this directory prevents your custom rules from being listed in the report. Because the Configuration Upgrade Tool does not modify your custom rules, there is no reason to compare your custom rules in the target configuration.
8. Navigate to the `bin` directory of your target ClaimCenter 6.0.8 version.
9. Enter the following command:

```
gwcc regen-rulereport
```


This command creates a rule repository report XML file in `build/rules`. There is a slight change to the path between the versions.
10. Append the target version number to the XML file name.
11. Restore the `modules/configuration/config/rules` directory to the target version.
12. Open both rule report XML files in a merge tool. You do not merge base rules using the rule repository reports. However, looking at changes that Guidewire has made to the base rules can help you determine the types of changes you must make in your custom rules.

In your merge tool, disable whitespace differences and comments to reduce the amount of inconsequential differences shown between rules.

Update custom rules using Studio. Studio does not compare your rules directly with target rules. However, Studio provides powerful Gosu editing features not available in a standard text editor that can alert you to issues.

When you have finished updating all of your custom rules, delete the ClaimCenter 6.0.8 rules directory from `modules/configuration/config/rules/rules`.

The ClaimCenter 6.0.8 default rules are enabled because some features depend on these rules.

Rules Required for Key Features

Some rules provided with ClaimCenter must be active for certain functionality to work. These rules are listed below by the feature for which they are required. Review these rules and determine which of the applicable functionality you want in your implementation.

Contact Automatic Synchronization

CCL02000 - Suspend AutoSync for Related Contacts

COP01000 - Update Check Address

Catastrophe Bulk Association

CPU09000 - Related to Catastrophe

CPU13000 - Catastrophe History

CLV10000 - Catastrophe

Claim Health Metrics

CPU18000 - Update Claim Health

EPU07000 - Update Claim Health

TPU07000 - Update Claim Health

Deductible Handling

EPU04000 - Update Deductible On Updated Exposure Coverage

EPU05000 - Update Deductible On Updated Coverage Deductible

EPU06000 - Stop Closing Of Exposure With Unpaid Deductible

TPU06000 - Unlink Deductible After Check Denial

Reinsurance

TPU04000 - Reinsurance

ISO Validation

CLV09000 - ISO Validation

Importing Translation Properties

During the upgrade from ClaimCenter 4.0 to 5.0, the Configuration Upgrade Tool moves localization information for display keys and typecodes into a `modules/configuration/config/locale/locale/translation.properties` file. See “Localizing Typelists” on page 183. Import `translation.properties` using

Studio to make the localized typecodes and display keys available.

IMPORTANT Before you can import localized typecodes and display keys for a locale, the locale must be defined in `localization.xml` and the language defined in the `LanguageType` typelist. This is most likely already configured if you already have localized typecodes to import for a locale.

To import localized typecodes and display properties

1. Open Studio from your ClaimCenter 6.0.8 environment by entering the following command from `ClaimCenter\bin`:

```
gwcc studio
```
 2. In the Studio Resources pane, right-click **configuration** → **Typelists**.
 3. Select **Import translation file** → *locale*, where *locale* is the name of the locale for which you want to import localized typecodes. A file browser window opens.
 4. Use the file browser window to navigate to `modules/configuration/config/locale/locale/`.
 5. Select the `translation.properties` file.
 6. Click **Open**. Studio imports the localized typecodes and display keys. Studio also copies localized display keys to a `modules/configuration/config/locale/locale/display.properties` file.
- After you import the localized typecodes and display keys, you can view them in Studio.

To view a localized typecode

1. Open Studio from your ClaimCenter 6.0.8 environment by entering the following command from `ClaimCenter/bin`:

```
gwcc studio
```
2. In the Studio Resources pane, expand **configuration** → **Typelists**.
3. Double-click the typelist that contains the localized typecode.
4. On the **Codes** tab, select the typecode.
5. Click the **Localizations** tab. Any localized values for the typecode are listed on this tab.

To view a localized display key

1. Open Studio from your ClaimCenter 6.0.8 environment by entering the following command from `ClaimCenter/bin`:

```
gwcc studio
```
2. In the Studio Resources pane, click **configuration** → **Display Keys**. The **Display Keys** tab opens.
3. Click a display key.
4. Select the locale from the drop-down menu. If there is a value defined for the selected locale, Studio displays the localized value below the drop-down.

Translating New Display Properties and Typecodes

ClaimCenter 6.0.8 adds new display properties and typecodes. If you have defined additional locales, export these new display properties and typecodes to a file, define localized values, and reimport the localized values. If you do not have additional locales defined in your ClaimCenter environment, skip this procedure.

To localize new display properties and typecodes

1. Open Studio from your ClaimCenter 6.0.8 environment by entering the following command from ClaimCenter\bin:

```
gwcc studio
```
 2. In the Studio Resources pane, right-click **configuration** → **Typelists**.
 3. Select **Export translation file** → **English (US) to locale**, where *locale* is the name of the locale for which you want to translate the new display properties and typecodes. A file browser window opens.
 4. Choose a location and name for the exported file.
 5. Open the file in a text editor. The first section of the file lists display properties and typecodes that have a localized value. The second section lists display properties and typecodes that do not have a localized value.
 6. Specify localized values for the untranslated properties.
 7. Save the updated file.
 8. In the Studio Resources pane, right-click **configuration** → **Typelists**.
 9. Select **Import translation file** → *locale*, where *locale* is the name of the locale for which you want to import the localized display properties and typecodes. A file browser window opens.
 10. Use the file browser window to navigate to the file containing translated display properties and typecodes.
 11. Click **Open**. Studio imports the localized typecodes and display keys. Studio also copies localized display keys to a `modules/configuration/config/locale/locale/display.properties` file.
- After you import the localized typecodes and display keys, you can view them in Studio.

Modifying PCF files, Rules and Libraries for Unused Contact Subtypes

You might not want to use all of the contact subtypes provided with ClaimCenter. However, contact subtypes are referenced within PCF files (particularly in modal pages of the address book tab and ContactCenter), rules, and libraries. Therefore, Guidewire recommends that you do not remove the unused contact subtypes. Instead, modify PCF files, rules and libraries that reference either the specific unused subtypes or the `ContactSubtype` entity itself.

To find contact subtype references

1. Open Guidewire Studio using the `gwcc studio` command in ClaimCenter/bin.
2. Right-click **Rule Sets, Classes, or Page Configuration (PCF)** and select **Find in Path**.
3. For **Text to find**, enter `ContactSubtype`.
4. Click **Find**.
5. Repeat this procedure, searching for unused contact subtype names instead of `ContactSubtype`. For example, if you are not using the `Adjudicator` contact subtype, search for `Adjudicator` to see which files, rules and libraries reference this unused subtype.

After you have found all references to the `ContactSubtype` entity and the specific unused contact subtypes, review each case to determine how to proceed.

If the usage is in a range input, as in `AddressBookSearchDV.pcf`, use a filter to exclude the unused contact subtypes from the range input. Alternatively, you can set the filter to only include the contact subtypes that you want to use.

If a menu item uses the contact subtype, as in `AddressBookMenuActions.pcf`, remove the menu item to prevent users from performing actions with the unused contact subtype.

Creating Modal PCF files for Custom Lines of Business

Note: This step is only required if you have one or more custom lines of business in addition to the standard set provided with ClaimCenter. The standard set includes Auto, General Liability, Property and Workers' Compensation. If you do not have custom lines of business, skip this topic.

ClaimCenter 5.0 and newer uses the modal PCF `NewClaimPolicyGeneralPanelSet` to display details of an unverified policy on the first page of the new claim wizard through the `NewClaimWizard_PolicyGeneralScreen` PCF. The `NewClaimPolicyGeneralPanelSet` is modal by loss type. If ClaimCenter can not locate a version of `NewClaimPolicyGeneralPanelSet` for each loss type, ClaimCenter throws an exception at run time.

If your configuration includes custom lines of business, create a modal version of `NewClaimPolicyGeneralPanelSet` for each custom loss type that you have.

Using Guidewire Studio, view existing modal versions of `NewClaimPolicyGeneralPanelSet`, such as `NewClaimPolicyGeneralPanelSet.Auto.pcf` as a guide to creating the new version. These files can be viewed through Guidewire Studio under **Resources** → **configuration** → **Page Configuration (PCF)** → **claim** → **FNOL**. Set the mode of the new `NewClaimPolicyGeneralPanelSet` to the code of the custom loss type.

Upgrading Assignment Rules

All assignment methods have been deprecated and replaced by methods with the same name, but a different signature. While upgrading these rules with the new methods, also note that Studio does not use its auto-complete feature on deprecated methods.

All assignment methods have been refactored so that they can be called from outside of the assignment engine. All old assignment method signatures have been deprecated. The method names are the same but the parameters have changed. There are also new assignment methods introduced in ClaimCenter 5.0.

The updated methods require the setting of `currentgroup`. To get the current group, use the following method:

```
AssignmentEngineUtil.getCurrentGroupFromES()
```

For more information, see “Assignment in ClaimCenter” on page 97 in the *Rules Guide*.

Upgrading ISO Rules

Due to significant changes to the ISO interface, legacy ClaimCenter rules pertaining to ISO do not function in ClaimCenter 6.0.8. To upgrade ISO rules, please use the target (6.0.8) rules as the basis for rewriting your own.

Upgrading ContactCenter Rules

After upgrading ContactCenter, upgrade rules before trying to create any contacts on the upgraded ContactCenter environment. Otherwise, ContactCenter fails to create the contact. One such rule that you must upgrade is `rules.V.ABContactValidationRules.DefaultValidationRules.RequirePastDOB`. You must change `Libraries.Date` in the action of this rule to `gw.api.util.DateUtil`. Examine, and if necessary modify, each rule in Studio to use syntax validation to ensure the rule is valid after upgrade.

Converting Velocity Templates to Gosu Templates

Guidewire has replaced Velocity templates with Gosu templates. If you have customized templates, you must manually convert these templates to Gosu. This topic outlines steps necessary to upgrade customized templates and maps old functionality to new.

The default templates for Claim and Check duplicate searches have been rewritten. If you are using the default duplicate search templates without modification, you do not need to do anything.

IMPORTANT If you have customized velocity templates, you must manually convert these templates to Gosu. Your customizations to the default templates appear in the Configuration Upgrade Tool under `config/templates/duplicate-search/`. You can compare changes you have made to the default version using a diff tool for reference purposes only.

New Location for Duplicate Search Templates

The duplicate search templates have been renamed and given a new extension. Gosu templates are now first class objects which live in the Gosu classes package structure.

Old Velocity template	New Gosu template	Description
<code>check.vm</code>	<code>gw.duplicaterearch.DuplicateCheckSearchTemplate.gst</code>	Duplicate Check search
<code>claim.vm</code>	<code>gw.duplicaterearch.DuplicateClaimSearchTemplate.gst</code>	Duplicate Claim search

You can access the Gosu duplicate search templates for ClaimCenter 6.0.8 using Guidewire Studio. Open **Configuration** → **Classes** → **gw** → **duplicaterearch**.

Parameters

The `DuplicateCheckSearchTemplate` takes three parameters:

- `DuplicateSearchHelper`, which provides utilities for SQL construction.
- the `Check` to search for duplicates.
- a `checkBeingCloned` parameter. If the `Check` is a clone of an existing check, then this parameter contains the existing `Check`. The search avoids returning the existing `Check` or any of its recurrences as a duplicate. Otherwise, `checkBeingCloned` is `null`.

These parameters differ from the original `check.vm` Velocity template. The original template took a couple of additional parameters: `PayeeTaxIds` and `CheckIdsToIgnore`, a possibly empty list of check ids that are not considered duplicates. The values that used to be in these parameters are now calculated in Gosu within the template, so the logic can be modified if necessary.

The `DuplicateClaimSearchTemplate` takes just two parameters:

- `DuplicateSearchHelper`, which provides utilities for SQL construction.
- the `Claim` to search for duplicates.

This closely mirrors the original parameters to the old `claim.vm` Velocity template, except that the `DuplicateSearchHelper` has changed a little, as described later in this topic.

Key Language Differences

The following table shows some key language differences between Velocity and Gosu.

Area	Velocity	Gosu
Comments	<pre> <code>/* This is a comment. */</code> </pre>	<pre> <code>/* This is a comment that spans multiple lines. */ and // This is a single-line comment</code> </pre>
Initializing a variable	<code>#set(\$myVar = 123)</code>	<code>var myVar = 123</code>
Modifying a variable	<code>#set(\$myVar = "new Value")</code>	<code>myVar = "new Value"</code>
Conditional expressions	<p>Conditional expressions that evaluate to null or false are equivalent.</p> <p>In the following Velocity example, the if block is only executed if \$myCondition evaluates to true.</p> <pre> <code>#if (\$myCondition) ... #endif</code> </pre> <p>The variable \$myCondition is evaluated to determine whether it is true. This happens under one of two circumstances:</p> <ul style="list-style-type: none"> \$myCondition is a Boolean that has a true value. the value is not null. The condition can be a non-Boolean object that is treated as true if not null and false if null. 	<p>The condition has to be a Boolean or the template does not compile. It is possible for a Boolean to be null, in which case it is treated as false.</p>
If conditional block	<pre> <code>#if (\$myCondition) ... #end</code> </pre>	<pre> <code>if (myCondition) { ... }</code> </pre>
If-else conditional block	<pre> <code>#if (\$myCondition) ... #else ... #end</code> </pre>	<pre> <code>if (myCondition) { ... } else { ... }</code> </pre>

VelocityUtil to DuplicateSearchHelper Transition Table

The support class `com.guidewire.cc.domain.duplicatesearch.VelocityUtil` is removed and superseded by `gw.api.util.DuplicateSearchHelper`. Some methods from `VelocityUtil` have been ported directly over to `DuplicateSearchHelper` and function identically. Other methods have been slightly changed. Some have been removed because they can be replaced by an equivalent Gosu expression or method call. The following table outlines these changes and equivalences. These methods are described in Studio. Use CTRL + Q in Studio to get details.

VelocityUtil Method	Status	Equivalence
<code>public String makeParam(Object value)</code>	unchanged	Identical method and signature on <code>DuplicateSearchHelper</code> .
<code>public String makeParam(String beanPath, Object value)</code>	unchanged	Identical method and signature on <code>DuplicateSearchHelper</code> .
<code>public String makeParam(IEntityPropertyInfo propertyInfo, Object value)</code>	unchanged	Identical method and signature on <code>DuplicateSearchHelper</code> .

VelocityUtil Method	Status	Equivalence
<code>public DateTimeUtil getDate()</code>	deleted	Use <code>gw.api.util.DateUtil</code> instead.
<code>public SQLUtil getSql()</code>	deleted	All methods on <code>SQLUtil</code> are now directly available on <code>DuplicateSearchHelper</code> .
<code>public int getArrayLength(Object[] array)</code>	deleted	Use <code>.length</code> property on array object.
<code>public Object getTypeCodeId(String typeName, String typeCode)</code>	modified	Replaced by <code>DuplicateSearchHelper.getTypeCodeId(TypeKey key)</code> .
<code>IEntityType getEntityIntrinsicType(String name)</code>	deleted	Use Gosu operator <code>typeof</code> .
<code>String escapeSqlString(String str)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).
<code>String getSQLStringValue(Object value)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).
<code>String compareIfEqualTo(Object value)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).
<code>String compareIfEqualToLowerCase(Object value)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).
<code>String format(String input, Object[] args)</code>	added	Used to be on <code>SQLUtil</code> (<code>getSql()</code>).

One specific example line to change is the following:

Velocity version:

```
VelocityUtil.getEntityIntrinsicType("Person").isAssignableFrom(myClaim.Insured)
```

Gosu version:

```
myClaim.Insured typeis Person
```

The original Velocity version of the line is included in the default duplicate search templates.

Validating the ClaimCenter 6.0.8 Configuration

This topic includes procedures to validate the upgraded configuration.

Using Studio to Validate Files

You can use Studio to validate files and rules without having to start ClaimCenter. Do not start ClaimCenter at this point. Studio can run without connecting to the application server.

To fix all problems in classes and enhancements including libraries, as well as PCF files and typelists:

1. Run the **Validate** option in the Studio **Tools** menu. This identifies all incorrect Gosu syntax in all Studio resources, issuing either a warning or an error.
2. Correct all identified errors with Studio. You can defer fixing warnings.

Starting ClaimCenter and Resolving Errors

IMPORTANT In the process described in this section, do not point the ClaimCenter server at a production database. The goal of this process is to test the configuration upgrade. Create an empty database account and point ClaimCenter to this account for this process. See “Configuring the Database” on page 20 in the *Installation Guide* and “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Upon starting the server for the first time, you might receive errors that prevent the server from starting. In general, fixing errors and starting the server is an iterative process that involves:

1. Start the server for the target configuration.

ClaimCenter encounters a configuration error and shuts down.

2. Copy the error message to a log file.

3. Locate the configuration causing the error - for example, a line of code in a PCF.

4. Comment out the offending line.

After the server starts successfully, look at the log and start solving errors and introducing solutions into the environment. Assign errors to developers on your team.

5. Copy the commented file to the test bed for later analysis.

6. Begin again with step 1. Continue until the server starts successfully.

When the server starts successfully, resolve any remaining issues in the configuration that caused startup errors. Attempt to resolve each error individually and start the server to see if the fix worked.

Building and Deploying ClaimCenter 6.0.8

After you apply and validate an upgrade to the configuration environment, rebuild and redeploy ClaimCenter. Before you begin, make sure you have carefully prepared for this step. In particular, make sure you have updated your infrastructure appropriately.

Review this topic and then rebuild and redeploy ClaimCenter to the application server. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide* of the target version for instructions.

WARNING Do not yet start ClaimCenter. Only package the application file and deploy it to the application server. Starting ClaimCenter begins the database upgrade.

If you have multiple Guidewire products to upgrade, such as ContactCenter, upgrade, build, and deploy each individually before attempting to re-integrate them.

The Build Environment

With the exception of the database configuration, the first time you start the application server use the unmodified `config.xml` and `logging.properties` files provided with the target configuration. After the server starts successfully, you can merge in specific configurations of these files.

If you are using a `build.xml` file to run the `ClaimCenter/bin` build tasks, ensure that it is updated correctly for the target infrastructure. You might encounter problems running build scripts if the data dictionary generation fails due to a PCF validation error. However, this dictionary does not report these errors.

If you encounter build failures due to data dictionary generation, you can comment out this dictionary generation. Then, as you start the server, it reports any PCF configuration errors. After you have corrected PCF configurations, un-comment the dictionary generation and rebuild the application file.

Preserving JAR Files

Place custom JAR files in the `/config/lib` directory. The `build-war`, `build-weblogic-ear`, or `build-websphere-ear` command and subsequent deployment of the packaged application copies the JAR file into the appropriate place for it to be accessed by ClaimCenter. JAR files in this location survive the upgrade process.

Integrating ContactCenter with ClaimCenter

If you started with ContactCenter integrated with a version of ClaimCenter before 5.0, use the following procedure to upgrade the integration:

1. From the ClaimCenter/bin directory, enter `gwcc studio` to launch Guidewire Studio.
2. In the **Resources** pane, under **Web Services**, select **abintegration**.
3. Set the URL to the address where ContactCenter is listening.
4. Click **Refresh**.
5. In the **Resources** pane, select **Plugins** → **gw** → **plugin** → **IAddressBookAdapter**.
6. Click **Remove**.
7. Click **Add**.
8. Select **Gosu**.
9. For the class, enter `gw.plugin.ccabintegration.impl.CCAddressBookPlugin`. This is the plugin provided with ClaimCenter. If you want to use a custom plugin, enter the class for that plugin instead. The class must be accessible to ClaimCenter.
10. Enter a username. The default username to use for the included plugin with the sample data set is `ClientAppCC`.
11. Enter a password for the user. The default password to use for the included plugin with the sample data set is `gw`.
12. Select **File** → **Save all changes**.

Advanced Authentication to ContactCenter

If in earlier releases you used the `DBAuthenticationPlugin` to hide authentication information when connecting from ClaimCenter to ContactCenter, then this procedure has changed as of ClaimCenter 5.0.1. Instead, you must use the similar `ABAuthenticationPlugin`. See “ABAuthenticationPlugin for ContactCenter Authentication” on page 246 in the *Integration Guide* for details.

Upgrading the ClaimCenter 4.0.x Database

This topic provides instructions for upgrading the ClaimCenter database to ClaimCenter 6.0.8.

Some procedures described in sections within this topic require the ClaimCenter toolkit in the customer configuration. If you do not see the `toolkit/bin` folder in the target configuration, run the `regen-toolkit` utility from the `bin` directory. After you have regenerated the toolkit, you can use the system tools on the current customer configuration. You do not need to generate the toolkit for the target configuration. Toolkit utilities in ClaimCenter 6.0.8 are in the `admin/bin` directory.

This topic includes:

- “Upgrading Administration Data for Testing” on page 234
- “Identifying Data Model Issues” on page 236
- “Verifying Batch Process and Work Queue Completion” on page 237
- “Purging Data Prior to Upgrade” on page 237
- “Validating the Database Schema” on page 238
- “Checking Database Consistency” on page 238
- “Creating a Data Distribution Report” on page 239
- “Generating Database Statistics” on page 240
- “Creating a Database Backup” on page 240
- “Updating Database Infrastructure” on page 240
- “Preparing the Database for Upgrade” on page 241
- “Handling Extensions” on page 241
- “Creating Extensions to Preserve Data” on page 243
- “Disabling Encryption for Upgrade Performance” on page 246
- “Setting Linguistic Search Collation for Oracle” on page 246
- “Reviewing Data Model Changes” on page 248

- “Using the IDatabaseUpgrade Plugin” on page 252
- “Disabling the Scheduler” on page 260
- “Suspending Message Destinations” on page 260
- “Configuring the Database Upgrade” on page 261
- “Starting the Server to Begin Automatic Database Upgrade” on page 264
- “Correcting Issues with Multiple Exposures on Incidents” on page 284
- “Viewing Detailed Database Upgrade Information” on page 286
- “Dropping Unused Columns on Oracle” on page 287
- “Setting Claim Descriptions” on page 287
- “Exporting Administration Data for Testing” on page 288
- “Final Steps After The Database Upgrade is Complete” on page 290

Upgrading Administration Data for Testing

You might want to create an upgraded administration data set for development and testing of rules and libraries with ClaimCenter 6.0.8. You can wait until the full database upgrade is complete and then export the administration data, as described in “Exporting Administration Data for Testing” on page 288. Or, you can upgrade only the administration data to have this data available earlier in the upgrade process. Use the procedure in this section to create an upgraded administration data set before upgrading the full database.

To upgrade administration data

1. Export administration data from your current (pre-upgrade) ClaimCenter production instance:
 - a. Log on to ClaimCenter as a user with the `viewadmin` and `soapadmin` permissions.
 - b. Click the **Administration** tab.
 - c. Choose **Import/Export Data**.
 - d. Select the **Export** tab.
 - e. Select **Admin** from the **Data to Export** dropdown.
 - f. Click **Export**. ClaimCenter exports an `admin.xml` file.
2. On a new pre-upgrade development environment based on your production configuration, create an empty version of `importfiles.txt` in the `modules/configuration/config/import/gen` directory.
3. Create empty versions of the following CSV files:
 - `activity-patterns.csv`
 - `authority-limits.csv`
 - `reportgroups.csv`
 - `roleprivileges.csv`
 - `rolereportprivileges.csv`Leave `roles.csv` as the original complete file.
4. Start the development environment server by opening a command prompt to `ClaimCenter/bin` and entering the following command:

```
gwcc dev-start
```
5. Import this administration data into the development environment.
 - a. Log on to ClaimCenter as a user with the `viewadmin` and `soapadmin` permissions.
 - b. Click the **Administration** tab.

- c. Choose **Import/Export Data**.
 - d. Select the **Import** tab.
 - e. Click **Browse....**
 - f. Select the `admin.xml` file that you exported in step 1.
 - g. Click **Open**.
6. Create a backup of the new development environment database.
7. Create a new database account for the development environment on a database management system supported by ClaimCenter 6.0.8. 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 <http://guidewire.custhelp.com>.
See “Configuring the Database” on page 20 in the *Installation Guide* for instructions to configure the database account.
8. Restore the backup of the database containing the imported administration data into the new database.
9. Connect your upgraded target ClaimCenter 6.0.8 configuration to the restored database.
10. Start the ClaimCenter 6.0.8 server to upgrade the database.
11. Export the upgraded administration data:
 - a. Start the ClaimCenter 6.0.8 server by navigating to `ClaimCenter/bin` and running the following command:
`gwcc dev-start`
 - b. Open a browser to ClaimCenter 6.0.8.
 - c. Log on as a user with the `viewadmin` and `soapadmin` permissions.
 - d. Click the **Administration** tab.
 - e. Choose **Import/Export Data**.
 - f. Select the **Export** tab.
 - g. For **Data to Export**, select **Admin**.
 - h. Click **Export**. Your browser will note that you are opening a file and will prompt you to save or download the file.
 - i. Select to download the `admin.xml` file.
12. Open `admin.xml`.
13. Modify each occurrence of `cc_systemTables` to `systemTables`.
14. Modify the `public-id` of the default organization from `<Organization public-id="default_data:1">` to `<Organization public-id="systemTables:1">`
15. Update all references to the default organization to the new `public-id` by changing each occurrence of `<Organization public-id="default_data:1"/>` to `<Organization public-id="systemTables:1"/>`.
16. Change the root group `public-id` from `<Group public-id="default_data:1">` to `<Group public-id="systemTables:1">`.
17. Change all references to the root group on group users and assignable queues by changing each occurrence of `<Group public-id="default_data:1"/>` to `<Group public-id="systemTables:1"/>`.
18. Change all references to the root group on child groups by changing each occurrence of `<Parent public-id="default_data:1"/>` to `<Parent public-id="systemTables:1"/>`.

19. Save changes to `admin.xml`. You can now import this upgraded administration data into your ClaimCenter 6.0.8 development environments.

Identifying Data Model Issues

Before you upgrade a production database, identify issues with the data model by running the database upgrade on an empty database. This process does not identify all possible issues. The database upgrade does not detect issues caused by specific data in your production database. Instead, this procedure identifies issues with the data model.

Complete the following procedure to identify data model issues, and correct any issues. Then follow the full list of procedures in this topic to upgrade a production database. This list begins with “Verifying Batch Process and Work Queue Completion” on page 237 and finishes with “Final Steps After The Database Upgrade is Complete” on page 290.

To identify data model issues

1. Create an empty schema of your starting version database. You can do this in a development environment by pointing the development ClaimCenter installation at an empty schema and starting the ClaimCenter server. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.
2. Complete the configuration upgrade for data model files in your starting version, according to the instructions in the previous topic. You do not need to complete the merge process for all files. See also “Handling Extensions” on page 241.
3. Create extensions to preserve data if applicable. Review “Creating Extensions to Preserve Data” on page 243.
4. Configure your upgraded development environment to point to the database account containing the empty schema of your old version. See “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.
5. Start the ClaimCenter server in your upgraded development environment. The server performs the database upgrade to ClaimCenter 6.0.8. See “Starting the Server to Begin Automatic Database Upgrade” on page 264.
6. Check for errors reported during the upgrade process. Resolve any issues before upgrading your production database. These errors can be issues detected by the database upgrade version checks or issues with the domain and admin graphs. ClaimCenter uses the domain and admin graphs for a number of features including archiving and purging. ClaimCenter 6.0.8 performs a number of graph validation checks when you start the server. The ClaimCenter server does not start if certain graph validation checks fail. In prior versions, graph validation errors did not prevent the server from starting.

See “Graph Validation Checks” on page 64 in the *System Administration Guide* for a description of the graph validation checks that ClaimCenter performs.

See “The Domain Graph” on page 279 in the *Configuration Guide* for a description of the domain and admin graphs. That topic includes information about configuring the data model properly to maintain graphs suitable for ClaimCenter.

You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. For more information, see “Running Custom SQL” on page 253.

If ClaimCenter detects circular references when the server starts, you can use the `IDatabaseUpgrade` plugin to convert foreign keys to edge foreign keys during the major version database upgrade. See “Eliminating Circular References” on page 255.

Verifying Batch Process and Work Queue Completion

All batch processes and distributed work queues must complete before beginning the upgrade. Check the status of batch processes and work queues in your current production environment.

To check the status of batch processes and work queues

1. Log in to ClaimCenter as the superuser.
2. Press **Alt + Shift + T**. ClaimCenter displays the **Server Tools** tab.
3. Click **Batch Process Info**.
4. Select **Any** from the **Processes** drop-down filter.
5. Click **Refresh**.
6. Check the **Status** column for each batch process listed. This list also includes batch processes that are writers for distributed work queues. If any of the batch processes have a **Status** of **Active**, wait for the batch process to complete before continuing with the upgrade.

Purging Data Prior to Upgrade

This topic includes recommendations for purging certain types of data from the database prior to upgrade. Removing unused records can improve the performance of the database upgrade and ClaimCenter.

Purging Old Messages from the Database

Purge completed inactive messages before upgrading the database. Doing so reduces the complexity of the database upgrade. Use the following command from the current (pre-upgrade) customer configuration `toolkit/bin` directory to purge completed messages:

```
messaging_tools -password password -server http://server:port/instance -purge MM/DD/YY
```

Replace `MM/DD/YY` with a date. This tool deletes completed messages that are older than that date. Periodically use this command to purge old messages to prevent the database from growing unnecessarily.

Or, you can use the web service API `IMessageToolsAPI.purgeCompletedMessages(Calendar date)`.

Always perform this message purge on the database before starting ClaimCenter so that the database upgrade does not attempt to convert those rows.

You cannot resend old messages after the upgrade. This is because integrations change and the message payload might be different. It is important that messages that have failed or not yet been consumed finish prior to upgrading.

After you purge completed inactive messages, reorganize the `cc_MessageHistory` table. You might also want to rebuild any indexes on the table. Contact Guidewire Support if you need assistance.

Purging Orphaned Policies from the Database

Each claim has an associated policy record. In ClaimCenter 6.0 and prior versions, if a user refreshed the policy information on a claim, the old policy record remained in the database. If orphaned policies are consuming large amounts of database space, contact Guidewire Support for assistance removing the policies.

Purging Address Correction Records

The `cc_addresscorrection` table stores address corrections returned from geocoding systems. ClaimCenter does not display address corrections by default. However, you might have configured ClaimCenter to expose address corrections to allow users to make corrections.

If you have not configured ClaimCenter to expose address corrections, you can remove the address correction records by truncating the `cc_addresscorrection` table.

If you do have address corrections exposed, you can remove records that have been handled already.

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

Use the following SQL statements in the order shown to remove these items:

```
delete from cc_workflowlog
  where workflow in ( select id from cc_workflow
    where state = ( select id from cctl_workflowstate where typecode="completed" ))
delete from cc_workflowworkitem
  where workflowid in ( select id from cc_workflow
    where state = ( select id from cctl_workflowstate where typecode="completed" ))
delete from cc_workflow
  where state = ( select id from cctl_workflowstate where typecode="completed" )
```

Try this in a test environment before applying to your production database.

ClaimCenter 6.0 and higher 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 the problem of a large number of workflow log records causing performance issues. These work queues are described in “Purging Old Workflows and Workflow Logs” on page 57 in the *System Administration Guide*.

Validating the Database Schema

This validation detects the unlikely event that the data model defined by your configuration files has become out of sync with the database schema. While the pre-upgrade server is running, use the `system_tools` command in `admin/bin` of the customer configuration to verify the database schema:

```
system_tools -password password -verifydbschema -server servername:port/instance
```

Correct any validation problems in the database before proceeding. Contact Guidewire Support for assistance.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Checking Database Consistency

ClaimCenter has hundreds of internal database consistency checks. Before upgrading, run consistency checks by executing the following command from the `admin/bin` directory of the customer configuration.

```
system_tools -password password -checkdbconsistency -server servername:port/instance
```

This command takes a long time and could time out. If it does, run the command on subsets of the database instead of the entire database. See the “system_tools Command” on page 173 in the *System Administration Guide* for instructions.

You can also trigger database consistency checks to run at server startup by setting `checker=true` in the database block of `config.xml`.

```
<database name="ClaimCenterDatabase" driver="dbcp"
  dbtype="sqlserver" autoupgrade="false" checker="true">
  ...
</database>
```

Each time the server starts with `checker=true`, it will run database consistency checks. These checks might take a long time to run. The server is not available until the consistency checks are completed. Therefore, when you have finished, disable consistency checks for the next server startup by setting `checker=false`.

Run database consistency checks early in the upgrade project. Fix any consistency errors. Continue to periodically run consistency checks and resolve any issues so that your database is ready to upgrade when you begin the upgrade procedure. Consistency issues might take some time to resolve, so begin the process of running consistency checks and fixing issues early.

IMPORTANT Resolve consistency check errors prior to upgrade, especially those surrounding Financials and Incident creation. After resolving any data issues, run the consistency checks again to ensure the database is ready to be upgraded. Contact Guidewire Support for information on how to resolve any consistency issues.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Creating a Data Distribution Report

Generate a data distribution report for the database before an upgrade. Save the output of this report. Run the report again after the upgrade to ensure the distribution is still correct.

Guidewire is very interested in the data distribution of your databases. Guidewire uses these reports to better understand the nature of your database and to optimize ClaimCenter performance. Guidewire appreciates copies of your reports, both before and after upgrades.

You can also use this information to tune the application server cache. See “Understanding Application Server Caching” on page 70 in the *System Administration Guide*.

To create a database distribution report

1. In `config.xml`, set `<param name="EnableInternalDebugTools" value="true"/>`.
2. Start the ClaimCenter application server.
3. Log into ClaimCenter as an administrative user.
4. Type ALT + SHIFT + T while in any screen to reach the **Server Tools** page.
5. Choose **Info Pages** from the **Server Tools** tab.
6. Choose the **Data Distribution** page from the **Info Pages** dropdown.
7. Enter a reason for running the Data Distribution batch job in the **Description** field.
8. On this page, select the **Collect distributions for all tables** radio button and check all checkboxes to collect all distributions.
9. Push the **Submit Data Distribution Batch Job** button on this page to start the data collection.

10. Return to the **Data Distribution** page and push its **Refresh** button to see a list of all available reports. The batch job has completed when the **Available Data Distribution** list on the **Data Distribution** page includes your description.
11. Select the desired report and use the **Download** button to save it zipped to a text file. Unzip the file to view it.

Generating Database Statistics

To optimize the performance of the ClaimCenter database, it is a good idea to update database statistics on a regular basis. Both SQL Server and Oracle can use these statistics to optimize database queries. Before you upgrade, and before you go into production, update the database statistics by running the `maintenance_tools` command from `toolkit/bin` of the customer (pre-upgrade) configuration.

To generate database statistics

1. Run the `maintenance_tools` command to get the proper SQL statements for updating the statistics in ClaimCenter tables:

```
maintenance_tools -getdbstatisticsstatements -password password  
-server http://server:port/instance> db_stats.sql
```

2. Run the resulting SQL statements against the ClaimCenter database.
3. If using an Oracle database, the `maintenance_tools` command creates `db_stats.ora`. Review and then run these statements against the database.

You can configure a SQL Server instance to periodically update statistics using SQL. See your database documentation and “Configuring Database Statistics” on page 53 in the *System Administration Guide* for more information.

The database upgrade can take a long time, and has built-in statistics collection that help you see if any part of the upgrade is slow. Collect these statistics, and compare them to the statistics you collected before the upgrade. The `config.xml` file has parameters that control this statistics collection.

Following the database upgrade, run this command again from the `admin/bin` directory of the target (upgraded) configuration.

Creating a Database Backup

Prepare the environment so that you can make a total recovery of the original installation if you run into problems during the upgrade.

The first time you start the ClaimCenter server after running the upgrade tool, the server updates the database. During its work, the database upgrader minimizes the logging that it does. For these reasons, back up your database before starting an upgrade. Your pre-upgrade database might not be recoverable after an upgrade.

Updating Database Infrastructure

Before starting the upgrade, update database server software and operating systems as needed to meet the installation requirements of ClaimCenter 6.0.8. 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>.

For SQL Server, after you upgrade the database server software, run the following command to set the compatibility level:

```
ALTER DATABASE databaseName SET COMPATIBILITY_LEVEL = 100
```


Enabling SQL Server READ_COMMITTED_SNAPSHOT Option

ClaimCenter 6.0.8 requires the SQL Server parameter READ_COMMITTED_SNAPSHOT be set to on. During startup, ClaimCenter 6.0.8 checks that this parameter is set. ClaimCenter only starts if the parameter is set properly.

To set READ_COMMITTED_SNAPSHOT, run the following SQL statement from a system administrator-level account:

```
ALTER DATABASE dbname
SET READ_COMMITTED_SNAPSHOT ON
WITH ROLLBACK IMMEDIATE
GO
```

IMPORTANT The use of READ_COMMITTED_SNAPSHOT greatly increases resource requirements on the tempdb database. Set tempdb to grow in 10% increments, and provide sufficient disk space for tempdb to grow substantially. Performance can be improved if you dedicate separate I/O resources to tempdb.

Preparing the Database for Upgrade

This topic notes steps to prepare the database for the upgrade process.

Ensuring Adequate Free Space

The database upgrade requires significant free space. Make sure the database has at least 50% of the current database size available as free space.

Disabling Replication

Disable database replication during the database upgrade.

Assigning Default Tablespace (Oracle only)

Set the default tablespace for the database user to the one mapped to the logical tablespace OP in config.xml.

The database upgrade creates temporary tables during the upgrade without specifying the tablespace. If the Oracle database user was created without a default tablespace, Oracle by default creates the tables in the SYSTEM tablespace. The Guidewire database user is likely not to have the required quota permission on the SYSTEM tablespace. This results in an error of the type:

```
java.sql.SQLException: ORA-01950: no privileges on tablespace 'SYSTEM'
```

Even if the default tablespace is not SYSTEM, if the Guidewire database user does not have quota permission on the default tablespace, the temporary table creation during upgrade fails.

Using Proper Clock Settings

The config.xml file contains several parameters set to AM/PM times (for example, <param name="BusinessDayEnd" value="5:00 PM"/>). If your server uses a 24-hour clock, change these parameters to reflect the server clock (<param name="BusinessDayEnd" value="17:00"/>).

Handling Extensions

This topic discusses how to handle extensions during the upgrade to ClaimCenter 6.0.8.

Merging Extensions

ClaimCenter 6.0.8 stores extensions in `.eti` and `.etx` files. An `Entity.eti` file defines a new entity. An `Entity.etx` file defines extensions to an existing entity. During the configuration upgrade, an automated step generates `.eti` and `.etx` files from your current `extensions.xml`. The Configuration Upgrade Tool then compares these files against the extension files included with ClaimCenter 6.0.8.

Guidewire often adds indexes to entities in the target configuration to improve the performance of database queries in ClaimCenter 6.0.8. ClaimCenter requires some of these indexes. Guidewire adds required indexes to entity definitions in the data model. Other indexes are recommended for most installations but can be disabled if they negatively impact performance. Guidewire adds optional indexes to entity extensions so you can disable any if necessary.

For example, to improve the performance of the team group activities and activity calendar pages, Guidewire added `Activity.ClaimID` and `Activity.Priority` to the `activityccu4` index on `Activity`. Guidewire also moved this index from the data model to an extension.

Use the Configuration Upgrade Tool to resolve extensions files. When you merge your custom extensions with Guidewire changes, review each new index added by Guidewire. In most cases, include the new index in the merged extension file. You can modify or remove index definitions based on usage in your deployment.

Reviewing Custom Extensions

Generate and review the data dictionary for the target version to identify any custom extensions that are now obsolete due to Guidewire adding a similar field to the base ClaimCenter.

To generate the data dictionary

1. From the command line, navigate to the `bin` directory of the target version.
2. Run the command `gwcc regen-dictionary`.

This command generates the data and security dictionaries in the `build/dictionary` directory of the target version. To view the data dictionary, open `build/dictionary/data/index.html` in a web browser.

Compare the target version data dictionary with the version in your current environment. If you have extensions that are now available as base fields, consider migrating the data in those fields to the base version. Consider whether an extension is still on the appropriate entity. A new entity could be a more appropriate location for the extension. Review key data model changes that might impact your custom extensions.

If you change an extension location or migrate to a new base field, update any PCF, rule or library that references the extension to reference the new location.

Reconciling the Database with Custom Extensions

The extensions defined in `.eti` and `.etx` files (previously in `extensions.xml`) must match the physical database. Delete all physical columns in the database that are not part of the base installation or defined as extensions before starting the server.

No extensions are migrated from Exposure or Claim to Incident. If an Exposure and Incident have matching extensions, the database upgrade copies data from the Exposure to the Incident record. See “Exposure Fields Copied to Incident” on page 248. An extension that is not present on both the Exposure and Incident is not migrated. Review your current extensions on Exposure and Claim and determine if the extension is more appropriate on the Incident object. If you move any extensions, update your PCF files to reference the new location.

Creating Extensions to Preserve Data

The database upgrade removes some tables and columns. If you have data in a table or column that you want to preserve, create an extension as described in the sections within this topic.

The database upgrade looks for specific extensions, and moves data from the table or column to be deleted to the extension before deleting. If there is data in the table or column to be deleted, and no extension defined, an upgrade check reports the issue. The upgrader then stops before it makes any changes to the database.

If you create an extension to preserve data, update references to that data in rules, libraries, PCF files, and so forth to point to the new extension.

Deductible Fields

The database upgrade removes deductible-related fields from Claim, Transaction and Payment. See “Removing Unused Deductible Columns” on page 152.

If you want to preserve these fields, add their definitions as extensions to the data model, as described in this topic. If the upgrader finds these fields defined in the data model, it does not remove them or their data during the upgrade.

To preserve deductible fields

1. Copy the `Claim.etx` file from the `cc/config/extensions` folder to `configuration/config/extensions` in the target configuration. If you already have `Claim.etx` in `configuration/config/extensions`, skip this step.
2. Open `Claim.etx` in a text editor.
3. Comment out the sample extensions listed below by adding `<!--` before their definitions and `-->` after, as shown. Leave the index definitions beneath these definitions uncommented.

```
<!--
<column default="foo" desc="varchar(30) extension; default value of 'foo'" name="VarcharExt"
  nullok="true" type="varchar">
  <columnParam name="size" value="30"/>
</column>
<column default="12" desc="integer extension; default value of '12'" name="IntegerExt" nullok="true"
  type="integer"/>
<array arrayentity="GWExtArray" desc="Sample array extension, using an entity type defined only in the
  extensions file." name="GWExtArray"/>
<typekey desc="Sample TypeKey extension." name="TypeKeyExt" nullok="true" typelist="LossType"/>
-->
```

4. Add a definition for a `DeductibleStatus` typekey, as shown in bold:

```
<!-- Extension to Claim -->
<extension entityName="Claim">
  ...
  <typekey desc="Whether the deductible has been paid." name="DeductibleStatus"
    typelist="DeductibleStatus"/>
  ...
</extension>
```

5. Save and close `Claim.etx`.
6. Create a `Transaction.etx` file in `configuration/config/extensions`.
7. Open `Transaction.etx`.
8. Add a definition for a `DeductiblePaid` typekey column.

```
<!-- Extension to Transaction -->
<extension entityName="Transaction">
  <typekey desc="Whether the deductible has been paid." name="DeductiblePaid"
    typelist="YesNo"/>
</extension>
```

9. Save and close `Transaction.etx`.

10. Create a `Payment.etx` file in `configuration/config/extensions`.

11. Open `Payment.etx`.

12. Add definitions for the following columns:

- `DeductibleAmount`
- `DeductibleSubtracted`
- `NetAmount`
- `OriginalAmount`

For example:

```
<!-- Extension to Payment -->
<extension entityName="Payment">
  <column desc="Amount subtracted for deductible." name="DeductibleAmount" nullok="true"
    type="money"/>
  <column desc="Has the deductible been paid?" name="DeductibleSubtracted" nullok="true"
    type="bit"/>
  <column desc="Net amount to pay." name="NetAmount" nullok="true"
    type="money"/>
  <column desc="Original payment amount." name="OriginalAmount" nullok="true"
    type="money"/>
</extension>
```

13. Save and close `Payment.etx`.

Reinsurance Status

The database upgrade removes the `ReinsuranceStatus` typekey and references to it. See “Removing Reinsurance Status Typekey and References” on page 151.

If you have data in `Claim.ReinsuranceStatus` that you want to keep, do the following:

- Create a `ReinsuranceStatus` typekey extension on `Claim`.
- Add the `ReinsuranceStatus` typelist back to the ClaimCenter 6.0.8 configuration.

These procedures are described in this topic.

If the upgrader does not detect a `ReinsuranceStatus` typekey extension, and there is data in `Claim.ReinsuranceStatus`, a version check reports an error. If there is no data in `Claim.ReinsuranceStatus`, and the extension is not defined, the upgrader drops the column. If the upgrader does detect the extension, it does not drop the `Claim.ReinsuranceStatus` column.

To preserve reinsurance status

1. Copy the `Claim.etx` file from the `cc/config/extensions` folder to `configuration/config/extensions` in the target configuration. If you already have `Claim.etx` in `configuration/config/extensions`, skip this step.
2. Open `Claim.etx` in a text editor.
3. If using the copied `Claim.etx`, comment out the sample extensions listed below by adding `<!--` before their definitions and `-->` after, as shown. Leave the index definitions beneath these definitions uncommented.

```
<!--
<column default="foo" desc="varchar(30) extension; default value of 'foo'" name="VarcharExt"
  nullok="true" type="varchar">
  <columnParam name="size" value="30"/>
</column>
<column default="12" desc="integer extension; default value of '12'" name="IntegerExt" nullok="true"
  type="integer"/>
<array arrayentity="GWExtArray" desc="Sample array extension, using an entity type defined only in the
  extensions file." name="GWExtArray"/>
<typekey desc="Sample TypeKey extension." name="TypeKeyExt" nullok="true" typelist="LossType"/>
-->
```

4. Add a definition for a `ReinsuranceStatus` typekey, as shown in bold:

```
<!-- Extension to Claim -->
<extension entityName="Claim">
  ...
```

```

    <typekey desc="The reinsurance status for a claim." name="ReinsuranceStatus"
    typelist="ReinsuranceStatus"/>
    ...
  </extension>

```

5. Save and close Claim.etx.
6. Create a ReinsuranceStatus.tti file in configuration/config/extensions in the target configuration.
7. Open ReinsuranceStatus.tti in a text editor.
8. Add the following text to ReinsuranceStatus.tti.

```

<?xml version="1.0"?>
<typelist
  xmlns="http://guidewire.com/typelists"
  desc=""
  name="ReinsuranceStatus">
  <typecode
    code="Does_Not_Apply"
    desc="Reinsurance does not apply"
    name="Does not apply"
    priority="1"/>
  <typecode
    code="Evaluating"
    desc="Evaluating reinsurance potential"
    name="Evaluating"
    priority="2"/>
  <typecode
    code="Pursuing"
    desc="Pursuing reinsurance"
    name="Pursuing"
    priority="3"/>
</typelist>

```

9. Save and close ReinsuranceStatus.tti.

Physical Property

The database upgrade renames the PhysicalProperty entity to PolicyLocation. If you had defined extensions on PhysicalProperty, rename the extension file and update the entityName attribute.

If you had defined extensions on PhysicalProperty, after you run the configuration upgrade, you have a ClaimCenter/modules/configuration/config/extensions/PhysicalProperty.etx file. To preserve your extension data, rename this file PolicyLocation.etx. Then, open the file and change the entityName attribute from PhysicalProperty to PolicyLocation.

Employment Class

The database upgrade renames the EmploymentClass entity to ClassCode. If you had defined extensions on EmploymentClass, rename the extension file and update the entityName attribute.

If you had defined extensions on EmploymentClass, after you run the configuration upgrade, you have a ClaimCenter/modules/configuration/config/extensions/EmploymentClass.etx file. To preserve your extension data, rename this file ClassCode.etx. Then, open the file and change the entityName attribute from EmploymentClass to ClassCode.

Policy Property and Policy Vehicle

In ClaimCenter 6.0, the policy property and policy vehicle entities are risk unit subtypes. The database upgrade moves data from these entities to RiskUnit. If you had defined extensions on either entity, rename the extension file and update the entityName attribute.

If you had defined extensions on PolicyProperty, after you run the configuration upgrade, you have a ClaimCenter/modules/configuration/config/extensions/PolicyProperty.etx file. To preserve your extension data, rename this file PropertyRU.etx. Then, open the file and change the entityName attribute from PolicyProperty to PropertyRU.

If you had defined extensions on `PolicyVehicle`, after you run the configuration upgrade, you have a `ClaimCenter/modules/configuration/config/extensions/PolicyVehicle.etx` file. To preserve your extension data, rename this file `VehicleRU.etx`. Then, open the file and change the `entityName` attribute from `PolicyVehicle` to `VehicleRU`.

If the database upgrade detects extensions defined for the risk unit subtypes, it copies the extension data from the original entity to `RiskUnit`.

Disabling Encryption for Upgrade Performance

ClaimCenter 6.0.8 defines a column override for `Contact.TaxID`. This column override enables encryption on `Contact.TaxID`. The database upgrade encrypts the `TaxID` values one by one. This process slows the database upgrade, particularly for implementations with a large number of contacts defined. You can disable the encryption to improve performance of the database upgrade. Then, in a subsequent maintenance window, you can enable encryption for the `TaxID` field.

To disable encryption of the contact tax ID

1. Open Studio 6.0.8 by entering the following command from the ClaimCenter 6.0.8 `ClaimCenter\bin` folder:

```
gwcc studio
```

2. In the Studio **Resources** pane, open **Data Model Extensions** → **extensions**.

3. Double-click `Contact.etx`.

4. Delete the following text from `Contact.etx`:

```
<column-override  
  name="TaxID">  
  <columnParam  
    name="encryption"  
    value="true"/>  
</column-override>
```

5. Click **Yes** when Studio asks if you would like to edit `Contact.etx`. Studio moves the file to the configuration module.

6. Click **File** → **Save Changes**.

After the database upgrade, you can add the column override back to `Contact.etx`. The next time you start the server, ClaimCenter encrypts each `Contact.TaxID`.

Setting Linguistic Search Collation for Oracle

IMPORTANT This section applies for Oracle implementations only. On SQL Server, ClaimCenter uses the collation setting of the database server. Database searching on SQL Server is always case-insensitive, and obeys the rules of the Windows collation set on the database. This section does not apply for SQL Server implementations.

WARNING Oracle Java Virtual Machine (JVM) must be installed on all Oracle databases hosting ClaimCenter. The only exception is when the ClaimCenter application locale is English and you only require case-insensitive searches. Ensure that Oracle initialization parameter `java_pool_size` is set to a value of above 50 MB.

You can specify how you want ClaimCenter to collate search results. The strength attribute of the `LinguisticSearchCollation` element of `GWLocale` for the default locale in `localization.xml` specifies how ClaimCenter sorts search results. You can set the strength to `primary` or `secondary`.

With `LinguisticSearchCollation` strength set to `primary`, ClaimCenter searches results in a case-insensitive and accent-insensitive manner. ClaimCenter considers an accented character equal to the unaccented version of the character if the `LinguisticSearchStrength` for the default application locale is set to `primary`. For example, with `LinguisticSearchCollation` strength set to `primary`, ClaimCenter treats “Reneé”, “Renee”, “renee” and “renee” the same.

With `LinguisticSearchCollation` strength set to `secondary`, ClaimCenter searches results in a case-insensitive, accent-sensitive manner. ClaimCenter does not consider an accented character equal to the unaccented version of the character if the `LinguisticSearchCollation` strength for the default application locale is set to `secondary`. For example, with `LinguisticSearchCollation` strength set to `secondary`, a ClaimCenter search treats “Renee” and “renee” the same but treats “Reneé” and “renee” differently. By default, ClaimCenter uses a `LinguisticSearchCollation` strength of `secondary`, for case-insensitive, accent-sensitive searching.

For Oracle, the following rules apply:

- **Case:** All searches ignore the case of the letters, whether `LinguisticSearchCollation` strength is set to `primary` or `secondary`. “McGrath” equals “mcgrath”.
- **Punctuation:** Punctuation is always respected, and never ignored. “O'Reilly” does not equal “OReilly”.
- **Spaces:** Spaces are respected. “Hui Ping” does not equal “HuiPing”.
- **Accents:** An accented character is considered equal to the unaccented version of the character if `LinguisticSearchCollation` strength is set to `primary`. An accented character is not equal to the unaccented version if `LinguisticSearchCollation` strength is set to `secondary`.

Japanese only

- **Half Width/Full Width:** Searches under a Japanese locale always ignore this difference.
- **Small/Large Kana:** Japanese small/large letter differences are ignored only when `LinguisticSearchCollation` strength is set to `primary`, meaning accent-insensitive.
- **Katakana/Hiragana sensitivity:** Searches under a Japanese locale always ignore this difference.
- The long dash character is always ignored.
- Soundmarks (`` and °) are only ignored if `LinguisticSearchCollation` strength is set to `primary`.

German only

- Vowels with an umlaut compare equally to the same vowel followed by the letter e. Explicitly, “ä”, “ö”, “ü” are treated as equal to “ae”, “oe” and “ue”.
- The Eszett, or sharp-s, character “ß” is treated as equal to “ss”.

ClaimCenter populates denormalized values of searchable columns to support the search collation. For example, with `LinguisticSearchCollation` strength set to `primary`, ClaimCenter stores the value “Reneé”, “Renee”, “renee” and “renee” in a denormalized column as “renee”. With `LinguisticSearchCollation` strength set to `secondary`, ClaimCenter stores a denormalized value of “renee” for “Renee” or “renee” and stores “renee” for “Reneé” or “renee”. Japanese and German locales make additional changes when storing values in denormalized columns in order to conform to the rules listed previously for those locales.

Any time you change the `LinguisticSearchCollation` strength and restart the server, ClaimCenter repopulates the denormalized columns. Previous versions of ClaimCenter populated the denormalized columns with lowercase values for case-insensitive search, equivalent to setting `LinguisticSearchCollation` strength to `secondary`. If you set `LinguisticSearchCollation` strength to `primary`, ClaimCenter repopulates the denormalized columns, substituting any accented characters for their base equivalents. This process can take a long time, depending on the amount of data. Therefore, if you want to change `LinguisticSearchCollation` strength to `primary`, you might want to do so after the database upgrade. If you are concerned about the duration of the database upgrade, you can change your search collation settings after the upgrade. During a mainte-

nance period, change `LinguisticSearchCollation` strength to `primary` and restart the server to repopulate the denormalized columns.

For Japanese locales, the ClaimCenter database upgrade from a prior major version repopulates the denormalized columns regardless of the `LinguisticSearchCollation` strength value. ClaimCenter must repopulate the denormalized columns for Japanese locales to have search results obey the Japanese-only rules listed previously.

Reviewing Data Model Changes

Before you begin the database upgrade, review data model changes included in this section and in the release notes.

This topic includes:

- “Default Size of `mediumtext` Columns” on page 248
- “Exposure Fields Copied to Incident” on page 248
- “Policy Data Model Changes” on page 248
- “Injury Information Moved to New `InjuryIncident` Entity” on page 250
- “New `MessageHistory` Entity” on page 252
- “Address Entities Consolidated” on page 252

Default Size of `mediumtext` Columns

In particular, the default size of `mediumtext` columns has been reduced from 2000 to 1333 to accommodate Oracle databases using the UTF-8 character set. On a UTF-8 Oracle database, 2,000 UTF-8 characters could exceed Oracle's maximum column size of 4,000 bytes. One UTF-8 character requires up to three bytes of storage.

Changing the size of `mediumtext` columns is a slow process that can add significant time to the database upgrade. Furthermore, if you have data in any `mediumtext` column that exceeds 1333 characters, you must truncate that data to a maximum of 1333 characters to fit. Otherwise, the database upgrader throws an exception.

If you are not using Oracle with UTF-8, Guidewire strongly suggests that you keep the size of `mediumtext` columns at 2000. This avoids the column size conversion, reducing the time required to perform the database upgrade and does not require you to truncate data in `mediumtext` columns that exceeds 1333 characters. To keep the size of `mediumtext` columns at 2000, set `<MediumTextDataType>=2000` in the `FieldValidators` typelist.

Exposure Fields Copied to Incident

The database upgrade inserts a new incident record for each exposure of an incident type. The upgrade copies any field on an exposure to a matching field of the same name and type on the new incident, if a matching field exists. This includes simple extensions on the exposure that have a matching extension of the same name and type on the incident. The upgrade does not copy array or reverse foreign key extensions.

The upgrade writes the ID of the incident to `Exposure.Incident`.

Policy Data Model Changes

The policy data model has been updated from ClaimCenter 4.0. The following sections describe changes to entities and fields and new entities and fields associated with the policy data model.

Changed Policy Model Entities and Fields

The database upgrade modifies the following entities and fields related to the policy data model.

Entity	Description
PolicyProperty and PolicyVehicle	<p>The database upgrade merges the PolicyProperty and PolicyVehicle tables to RiskUnit, which has several subtypes. VehicleNumber and PropertyNumber become RUNumber. The combination of RUNumber and Subtype must be unique. RiskUnit also has a foreign key into the ClassCodes array on Policy and an array of Coverages.</p> <p>PolicyVehicle rows become VehicleRU rows. VehicleRU is a subtype of RiskUnit.</p> <p>PolicyProperty rows become some subtype of LocationBasedRU, determined by the following rules:</p> <ul style="list-style-type: none"> • If the policy is a workers' compensation policy (PolicyType with code value wc or comp), then the RiskUnit subtype for the policy is WCCovEmpRU. If there are several WCClassCodes on the original PhysicalProperty, those codes are moved to Policy and a WCCovEmpRU is created for each class code. • If the policy is not a workers' compensation policy and PolicyProperty.Property.BuildingNumber is null, then the RiskUnit subtype is PropertyRU. • If the policy is not a workers' compensation policy and PolicyProperty.Property.BuildingNumber is not null, then the RiskUnit subtype is BuildingRU.
VehicleCoverage.Vehicle	The upgrade converts VehicleCoverage.Vehicle to RUCoverage.RiskUnit. RUCoverage is supertype of VehicleCoverage.
PropertyCoverage.Property	The upgrade converts PropertyCoverage.Property to RUCoverage.RiskUnit. RUCoverage is supertype of PropertyCoverage.
Vehicle.Coverages	The upgrade moves Vehicle.Coverages to RiskUnit.RUCoverages.
PhysicalProperty	The upgrade replaces the PhysicalProperty table with a new table, PolicyLocation. Any extensions on PhysicalProperty are copied to PolicyLocation.
PhysicalProperty.BuildingNumber	The upgrade converts PhysicalProperty.BuildingNumber to an array of Building entities on PolicyLocation. The Building entity is introduced in ClaimCenter 5.0. If PhysicalProperty.BuildingNumber is populated, the upgrade creates a Building entity and adds it to the PolicyLocation.Buildings array.
PhysicalProperty.Location	When the upgrade copies the Location field from PhysicalProperty to PolicyLocation, it renames the field to PhysicalProperty.LocationNumber.
PhysicalProperty.WCClassCodes	The upgrade moves PhysicalProperty.WCClassCodes to Policy.ClassCodes. RiskUnit has a foreign key into the Policy.ClassCodes array. For each code in the WCClassCodes array, the upgrade create a RiskUnit and points it to the appropriate code on the policy and the PolicyLocation from which the code came.
PhysicalProperty.Coverages	The upgrade moves PhysicalProperty.Coverages to RiskUnit.RUCoverages.
EmploymentClass	The upgrade renames EmploymentClass to ClassCode. The upgrade adds a ClassCodes array to Policy and is meant to be more general than EmploymentClass, which was specific to workers' compensation claims.

New Policy Model Entities and Fields

These are completely new fields and entities related to the policy data model that have no analog in ClaimCenter 4.0.

Entity	Description
Policy.ClassCodes	All of the ClassCodes, formerly known as EmploymentClass, for the Policy. A RiskUnit can point into this array to indicate which ClassCode applies.
RUCoverage	New subtype of Coverage that adds a foreign key to RiskUnit. VehicleCoverage and PropertyCoverage are now subtypes of the RUCoverage type and lose their foreign keys to Vehicle and Property, respectively.

Entity	Description
PolicyLocation.PrimaryLocation	A Boolean indicating if the property is the primary one on the policy.
PolicyLocation.Policy	The policy that owns this PolicyLocation. All PolicyLocations linked from a RiskUnit are also stored in the array Policy.PolicyLocations.
Building	Provides information about a building. Buildings associated with a PolicyLocation are available as PolicyLocation.Buildings. Supercedes PhysicalProperty.BuildingNumber.
CovTerm	Encapsulates extra coverage information beyond the base limits fields on Coverage. A Coverage now includes an array of this additional terms and conditions information, Coverage.CovTerms.

Injury Information Moved to New InjuryIncident Entity

This applies for ClaimCenter upgrades from any version prior to 5.0.

Injury-related information, formerly part of Claim and Exposure in versions of ClaimCenter prior to 5.0, is now part of InjuryIncident. This change might require you to update customizations that used injury information from the Claim or Exposure. Specifically, the following fields are no longer a part of a Claim or Exposure:

Entity	Old Fields	Moved to InjuryIncident field
Claim	PrimaryInjury	Claim.ClaimInjuryIncident.GeneralInjuryType
	DetailedInjury	Claim.ClaimInjuryIncident.DetailedInjuryType
	PrimaryBodyPart	Claim.ClaimInjuryIncident.FirstBodyPart.PrimaryBodyPart (On first row in the OrderedBodyParts array)
	DetailedBodyPart	Claim.ClaimInjuryIncident.FirstBodyPart.DetailedBodyPart (On first row in the OrderedBodyParts array)
	MedicalTreatmentType	Claim.ClaimInjuryIncident.MedicalTreatmentType
	InjuryDescription	Claim.ClaimInjuryIncident.Description
	Severity	Claim.ClaimInjuryIncident.Severity
Exposure	LeadingInjury	Exposure.InjuryIncident.GeneralInjuryType
	WCLeadingInjury	
	MPLeadingInjury	
	DetailedInjury	Exposure.InjuryIncident.DetailedInjuryType
	WCDetailedInjury	
	MPDetailedInjury	
	PrimaryBodyPart	Exposure.InjuryIncident.FirstBodyPart.PrimaryBodyPart (In first BodyPartDetails in the OrderedBodyParts array)*
	WCPPrimaryBodyPart	
	MPPPrimaryBodyPart	
	DetailedBodyPart	Exposure.InjuryIncident.FirstBodyPart.DetailedBodyPart
	WCDetailedBodyPart	
	MPDetailedBodyPart	
	MedicalTreatment	Exposure.InjuryIncident.MedicalTreatmentType
	WCMedicalTreatment	
	MPMedicalTreatment	
	Severity	Exposure.Incident.Severity
	Impairment	Exposure.InjuryIncident.Impairment
	LWImpairment	
	LostWages	Exposure.InjuryIncident.LostWages
	PrimaryDoctor (ClaimContact role)	Exposure.InjuryIncident.PrimaryDoctor

* - if the incident was created by upgrading from a previous version of ClaimCenter.

Use the previous table to update field references in PCF files and rules.

ClaimInjuryIncident

There were some injury-related fields on `Claim` akin to those on `Exposure`, and they have also been moved to `InjuryIncident`. To store the values of these fields that might have been set on `Claim`, there is a special `InjuryIncident` called `ClaimInjuryIncident`. This incident is available as `Claim.ClaimInjuryIncident`. By default, this incident is only used by workers' compensation claims. But, `ClaimInjuryIncident` could be present on other claim types after upgrade if the injury-related fields on `Claim` had been set.

To access the `ClaimInjuryIncident` and its fields in PCF files, use the `Claim.ensureClaimInjuryIncident()` method. Use `Claim.ClaimInjuryIncident` to examine `ClaimInjuryIncident` fields in rules.

Fields removed from Exposure

The `LeadingInjury` and `DetailedInjury` fields have a 1:1 relationship with an `Incident`. So the following four fields are deleted:

- `WCLeadingInjury`
- `MPLeadingInjury`
- `WCDetailedInjury`
- `MPDetailedInjury`

Version checks make sure that these fields do not exist on the same exposure along with `LeadingInjury` and `DetailedInjury`. If the fields exist, manually clean the data before proceeding with the upgrade.

Additionally, the following six fields have been removed from `Exposure`:

- `WCPrimaryBodyPart`
- `MPPPrimaryBodyPart`
- `WCDetailedBodyPart`
- `MPDetailedBodyPart`
- `WCMedicalTreatment`
- `MPMedicalTreatment`

The data are not lost, however. They are represented in an array, which maintains the ordering of the body part details, for example, in a consistent manner.

Unchanged Fields

The following fields, mostly Medical Case Management-related, have remained unchanged. These fields are still attached to the `Claim` or `Exposure`.

- `FurtherTreatment`
- `hospital`
- `HospitalDate`
- `HospitalDays`
- `InjuredOnPremises`
- `InjuredRegularJob`
- `MedicalDiagnosis`
- `MedicalTreatment`
- `ShowMedicalFirstInfo`
- `doctor`
- `DrugsPrescribed`
- `injured`
- `MedicalContactStatus`
- `PhysTherapist`

For information about incident typelists, see "Incidents" on page 235 in the *Application Guide*.

New MessageHistory Entity

ClaimCenter 6.0.8 includes a new entity, MessageHistory.

If you have defined any extensions on the Message object, create the same extensions on the new MessageHistory object. The upgrade does not automatically apply Message extensions to MessageHistory.

Address Entities Consolidated

The entities around addressing have been consolidated. The following entities have all become part of a single Address entity:

- ABAddress
- NonPersistentAddress
- Geocodable

If you have extensions defined for ABAddress, apply the extensions to the Address entity instead.

Using the IDatabaseUpgrade Plugin

The IDatabaseUpgrade plugin interface provides hooks for custom code that you want to run during the database upgrade. You can use the IDatabaseUpgrade plugin to run custom SQL before and after the database upgrade. You can also use the IDatabaseUpgrade plugin to eliminate circular references in your data model by converting specific foreign keys to edge foreign keys.

WARNING The IDatabaseUpgrade plugin runs any time that the server determines that a database upgrade is necessary, whether for a major or minor version. Therefore, use conditionals within your preUpgrade and postUpgrade methods and get EdgeForeignKeyUpgrades property to control execution of your custom code. Test these conditionals thoroughly to ensure that the custom code you define within the methods runs appropriately. Remove or disable the IDatabaseUpgrade plugin once you have completed the upgrade.

The IDatabaseUpgrade plugin interface includes the following properties:

```
get BeforeUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
get AfterUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
```

These properties are for making custom upgrades to archived XML entities. This is only applicable to implementations that have enabled archiving. If you do not have archiving enabled, return an empty list for these properties.

```
override property get BeforeUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}

override property get AfterUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}
```

If you make custom data model changes within ClaimCenter and enable archiving, you can use these properties to define changes for the upgrader to make to the archived XML. See “Upgrading Archived Entities” on page 258.

Running Custom SQL

You can use the `IDatabaseUpgrade` plugin to run custom SQL before and after the database upgrade. For example, you might fix a consistency check failure issue, correct issues reported by version checks, or delete a custom extension that you are no longer using.

The `IDatabaseUpgrade` plugin interface contains method signatures for two methods that you must define in your plugin. These signatures are:

- `function preUpgrade(context : IUpgradeContext)`
- `function postUpgrade(context : IUpgradeContext)`

The `preUpgrade` method runs before version checks, so you can include any SQL to resolve version check issues in the `preUpgrade` method. You can also write SQL to resolve consistency check issues among other operations.

The `postUpgrade` method runs after the database upgrade version triggers. You can incorporate SQL in the `postUpgrade` method to move data into tables or columns that did not exist prior to upgrading.

The `IDatabaseUpgrade` interface also contains an `Iterable` property used for eliminating circular references. You must define this property. If you are not using this feature of the `IDatabaseUpgrade` plugin, you can define an empty `Iterable` as follows:

```
override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
    return {}
}
```

Any custom SQL that you write must be idempotent. That is, the SQL must be designed so that it can run against the database multiple times without corrupting data. The `IDatabaseUpgrade` plugin runs each time the database is upgraded, whether for a major or minor version of ClaimCenter or for a data model change such as a new extension. Check the version of the database in your `preUpgrade`, `postUpgrade` methods, and `get EdgeForeignKeyUpgrades` property to ensure that your custom SQL is executed appropriately. Guidewire recommends that you create a custom method to check the database version.

For ClaimCenter 5.0 you can determine the database major and minor version by opening `modules\cc\config\metadata\metadataproperties.xml` and checking the value of the `majorVersion` and `minorVersion` properties.

For 6.0 versions of ClaimCenter, the internal major and minor version numbers are listed in `modules\cc\config\metadata\metadata.properties`.

You can also determine the major and minor version of a ClaimCenter instance by reading the messages reported by the server during startup.

```
[java] serverName timestamp INFO Validating main database connections have all required properties
[java] serverName timestamp INFO Checking if existing database version is current...
[java] serverName timestamp INFO Datamodel version (major, minor, platform) is up to date.
```

This message reports the internal major, minor and platform version numbers. Only the major and minor version are of interest for writing custom SQL. Note that these are internal version numbers and do not match the commonly used external version number, such as 6.0.8.

To run custom SQL

1. Create a new Gosu class that implements `IDatabaseUpgrade`. If you already created a Gosu class to eliminate circular references, you can use the `preUpgrade` and `postUpgrade` methods of that class. The class you create must define the `preUpgrade` and `postUpgrade` methods and the `get EdgeForeignKeyUpgrades` property.

The following example demonstrates using the `preUpgrade` method to set `Claim.DeductibleStatus`, `Claim.ReinsuranceStatus` and `Claim.LossLocationID` to null:

```
package gw.myplugins.upgrade.impl

uses gw.plugin.upgrade.IDatabaseUpgrade
uses gw.plugin.upgrade.IUpgradeContext
uses java.lang.Iterable
uses gw.plugin.upgrade.EdgeForeignKeyUpgradeInfo
uses gw.api.archiving.upgrade.ArchivedDocumentUpgradeVersionTrigger

class DatabaseUpgradeImpl implements IDatabaseUpgrade {

    construct() {
    }

    // custom function to determine if DB version is prior to ClaimCenter 6.0 (DB version 8).
    // MajorVersion of CC 5.x is 7, MajorVersion for CC 6.x is 8

    private function isMajorVersionLessThan8(upgradeContext : IUpgradeContext) : boolean {

        //if null upgrade context is sent, return false and log reason
        if (upgradeContext == null) {
            //printing instead of logging because logs were not shown in console
            print("databaseupgrade.isMajorVersionLessThan8() - null upgradeContext sent")
            return false
        }

        //try to access major version, was throwing exception
        try {

            if (upgradeContext.CurrentMajorVersion < 8) {
                return true
            }

        } catch(e) {

            //swallow exception and return false
            print("databaseupgrade.isMajorVersionLessThan8() - accessing " +
                "upgradeContext.CurrentMajorVersion threw exception. Swallowing following exception" +
                "and returning false.")

            e.printStackTrace()

            return false
        }

        //All other conditions, return false
        return false
    }

    override function preUpgrade(upgradeContext : IUpgradeContext) {

        // first check the major version of the database to determine if scripts need to be run.
        if (isMajorVersionLessThan8(upgradeContext)) {

            // DB will be upgraded to CC6.0, execute pre-upgrade scripts
            upgradeContext.update("Set DeductibleStatus and ReinsuranceStatus to null",
                "UPDATE CC_CLAIM " +
                "SET DeductibleStatus = null, ReinsuranceStatus = null " +
                "WHERE DeductibleStatus is not null OR ReinsuranceStatus is not null")

            upgradeContext.update("Null out LossLocationID",
                "UPDATE CC_CLAIM " +
                "SET LossLocationID = null " +
                "WHERE LossLocationID is not null")

        }
    }

    override function postUpgrade(upgradeContext : IUpgradeContext) {

        // first check the major version of the database to determine if scripts need to be run.
        if (isMajorVersionLessThan8(upgradeContext)) {

            ///# todo: Implement

        }
    }

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
```

```

        // If not applicable, return empty iterable.
        return {};
    }

    override property get BeforeUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }

    override property get AfterUpgradeArchivedDocumentChanges() :
        List<ArchivedDocumentUpgradeVersionTrigger> {

        final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
            ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

        return list;
    }
}

```

2. Implement the `IDatabaseUpgrade` plugin with the new class. If you already implemented this class to eliminate circular references, skip this step.

- a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
- b. In the Studio **Resources** pane, expand **Plugins** → **gw** → **plugin** → **upgrade**.
- c. Right-click **IDatabaseUpgrade** and select **Implement**.
- d. On the **IDatabaseUpgrade** tab, click **Add...**
- e. Select **Gosu**.
- f. Enter your class, including the package.
- g. Select **File** → **Save Changes**.

When you start the server to perform the database upgrade from a prior major version, the upgrade calls the plugin and runs your custom `preUpgrade` and `postUpgrade` methods.

Eliminating Circular References

ClaimCenter 6.0.5 and newer enforces data model integrity more strictly than previous versions. ClaimCenter runs a series of checks on the data model during server startup. If any of these checks fails, ClaimCenter reports an error and does not start. One of these checks looks for circular references in the data model.

A circular reference exists when an entity points to an entity of the same type through one or more foreign keys. The circular reference can be simple, in which the entity has a foreign key directly to the same entity. Or, the reference can be more complex, in which an entity has a foreign key to another entity that directly or indirectly references the original entity.

To resolve circular references in the data model, use the `IDatabaseUpgrade` plugin to convert foreign keys causing circular references to edge foreign keys. This plugin enables you to specify the entity, existing foreign key and the edge foreign key to create. The plugin is called during the database upgrade from a prior major version.

The interface also contains an `Iterable` property used for eliminating circular references. You must define this property. If you are not using this feature of the `IDatabaseUpgrade` plugin, you can define an empty `Iterable` as follows:

```

    override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
        return {}
    }
}

```

The `IDatabaseUpgrade` plugin interface also contains method signatures for methods that you must define in your plugin. These signatures are:

- `function preUpgrade(context : IUpgradeContext)`
- `function postUpgrade(context : IUpgradeContext)`

If you do not want to run custom SQL before or after the upgrade, you can create these methods with no operations defined within them.

To convert foreign keys to edge foreign keys

1. Change the data model to replace the foreign key with an edge foreign key.
 - a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
 - b. In the Studio **Resources** pane, expand **Data Model Extensions** → **extensions**.
 - c. Double-click *EntityName.etc* to open the file defining the entity extension.
 - d. Change the `foreignKey` element to `edgeForeignKey`.
 - e. Add an `edgeTableName` attribute and set the value to a name for the edge table, such as `edgeTableName="entitynameedge"`.
 - f. Remove the `columnName` attribute.
 - g. If you want the contents of the edge table to be part of the domain graph, add the following subelement to `edgeForeignKey`:

```
<implementsEntity name="Extractable"/>
```

 Be sure to move the closing tag of the `edgeForeignKey` element if necessary to encapsulate the subelement.
 - h. Select **File** → **Save Changes**.
 - i. Repeat steps c through h for each foreign key that you want to convert to an edge foreign key.
2. Create a new Gosu class that implements `IDatabaseUpgrade`. If you already created a Gosu class to run custom SQL before or after the upgrade, you can use the `get EdgeForeignKeyUpgrades` property of that class. The class you create must define the `get EdgeForeignKeyUpgrades` property and the `preUpgrade` and `postUpgrade` methods. If you do not want to run custom SQL before or after the upgrade, you can create these methods with no operations defined within them. For example:

```
package gw.myplugins.upgrade.impl
uses gw.plugin.upgrade.IDatabaseUpgrade
uses gw.plugin.upgrade.IUpgradeContext
uses java.lang.Iterable
uses gw.plugin.upgrade.EdgeForeignKeyUpgradeInfo
uses gw.api.archiving.upgrade.ArchivedDocumentUpgradeVersionTrigger

class DatabaseUpgradeImpl implements IDatabaseUpgrade {

  construct() {
  }

  private var _upgradeContext : IUpgradeContext

  // custom function to determine if DB version is prior to ClaimCenter 6.0 (DB version 8).
  // MajorVersion of CC 5.x is 7, MajorVersion for CC 6.x is 8

  private function isMajorVersionLessThan8(upgradeContext : IUpgradeContext) : boolean {
    _upgradeContext = upgradeContext

    //if null upgrade context is sent, return false and log reason
    if (upgradeContext == null) {
      //printing instead of logging because logs were not shown in console
      print("databaseupgrade.isMajorVersionLessThan8() - null upgradeContext sent")
      return false
    }

    //try to access major version, was throwing exception
```



```

try {
    if (upgradeContext.CurrentMajorVersion < 8) {
        return true
    }
} catch(e) {

    //swallow exception and return false
    print("databaseupgrade.isMajorVersionLessThan8() - accessing" +
        "upgradeContext.CurrentMajorVersion threw exception. Swallowing following exception" +
        "and returning false.")

    e.printStackTrace()

    return false
}

//All other conditions, return false
return false
}

override property get EdgeForeignKeyUpgrades() : Iterable<EdgeForeignKeyUpgradeInfo> {
    // Check the major version of the database to determine which edge foreign keys must be converted.
    // MajorVersion of CC5.x is 7, MajorVersion for CC6.x is 8
    if (isMajorVersionLessThan8(_upgradeContext)) {

        return {
            // add these as necessary
            // new EdgeForeignKeyUpgradeInfo("foreignKeyColumn", entityName, "edgeForeignKey")
        }
    }
    else {
        return {}
    }
}

override function preUpgrade(p0 : IUpgradeContext) {
}

override function postUpgrade(p0 : IUpgradeContext) {
}

override property get BeforeUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}

override property get AfterUpgradeArchivedDocumentChanges() :
    List<ArchivedDocumentUpgradeVersionTrigger> {

    final ArrayList<ArchivedDocumentUpgradeVersionTrigger> list = new
        ArrayList<ArchivedDocumentUpgradeVersionTrigger>();

    return list;
}
}

```

Add a new `EdgeForeignKeyUpgradeInfo("foreignKeyColumn", entityName, "edgeForeignKey")` line to the `get EdgeForeignKeyUpgrades()` definition for each foreign key that you want to convert. Separate these lines with commas.

3. Implement the `IDatabaseUpgrade` plugin with the new class. If you already implemented this class to run custom SQL before or after the database upgrade, skip this step.
 - a. Start Guidewire Studio 6.0.8 by entering `gwcc studio` from the `ClaimCenter\bin` directory.
 - b. In the Studio Resources pane, expand **Plugins** → **gw** → **plugin** → **upgrade**.
 - c. Right-click `IDatabaseUpgrade` and select **Implement**.
 - d. On the `IDatabaseUpgrade` tab, click **Add...**
 - e. Select **Gosu**.

f. Enter your class, including the package.

g. Select **File** → **Save Changes**.

When you start the server to perform the database upgrade from a prior major version, the upgrade calls the plugin and converts the foreign keys to edge foreign keys.

Upgrading Archived Entities

ClaimCenter 6.0.5 and newer include a file-based archiving feature. If you implement archiving, and you make custom data model changes, then you can upgrade the archived XML using the `IDatabaseUpgrade` plugin.

Simple data model changes do not require a custom trigger. These include:

- Adding a new entity
- Updating denormalization columns
- Adding editable columns such as `updatetime`
- Adding new columns
- Changing the nullability of a column

More complex transformations or those that could result in loss of data require a version trigger. These include:

- changing a datatype (other than just length)
- migrating data from one table or column to another
- dropping a column
- dropping a table
- renaming a column
- renaming a table

ClaimCenter upgrades an archived entity as the entity is restored.

The `IDatabaseUpgrade` plugin interface includes the following properties:

```
get BeforeUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
get AfterUpgradeArchivedDocumentChanges() : List<ArchivedDocumentUpgradeVersionTrigger>
```

You can define custom `ArchivedDocumentUpgradeVersionTrigger` entities to modify archived XML. The `ArchivedDocumentUpgradeVersionTrigger` is an abstract class that you can extend to create your custom triggers.

Define the constructor of your custom `ArchivedDocumentUpgradeVersionTrigger` to call the constructor of the superclass and pass it a numeric value. For example:

```
construct() {
    super(171)
}
```

This numeric value is the extension version to which your trigger applies. If you run the upgrade against a database with a lower extension version, then your custom trigger is called. The current extension version is defined in `modules\cc\config\extensions\extensions.properties`.

Provide an override definition of the `get Description` property to return a `String` that describes the actions of your trigger.

Provide an override definition for the `execute` function to define the actions that you want your custom trigger to make on archived XML.

When the upgrade executes your custom trigger, it wraps each XML entity in an `IArchivedEntity` object. Each typekey is wrapped in an `IArchivedTypekey` object. The upgrade operates on a single XML document at a time.

`ArchivedDocumentUpgradeVersionTrigger` provides the following key operations:

- `getArchivedEntitySet(entityName : String)` – returns an `IArchivedEntitySet` object that contains all `IArchivedEntity` objects of the given type in the XML document.

`IArchivedEntitySet` provides the following key methods:

- `rename(newEntityName : String)` – renames all rows in the set to the new name.
- `delete()` – deletes all rows in the set.
- `search(predicate : Predicate<IArchivedEntity>)` – returns a List of `IArchivedEntity` objects that match the given predicate.
- `create(referenceInfo : String, properties : List<Pair<String, Object>>)` – returns a new `IArchivedEntity` with the given properties.

`IArchivedEntity` provides the following key methods:

- `delete()` – deletes just this row.
- `getPropertyValue(propertyName : String)` – returns the value of the property of the given name. If the property value is a reference to another entity, this method returns an `IArchivedEntity`.
- `move(newEntityName : String)` – moves this to a new entity type of the given name. The type is created if it did not exist. You must add any required properties to the type.
- `updatePropertyValue(propertyName : String, newValue : String)` – updates the property of the given name to the given value.
- `getArchivedTypekeySet(typekeyName : String)` – returns an `IArchivedTypekeySet` object that contains all `IArchivedTypekey` objects in the given typelist.
- `getArchivedTypekey(typelistName : String, code : String)` – Returns an `IArchivedTypekey` representing the typekey.

`IArchivedTypekey` provides the following key method:

- `delete()` – deletes the typekey from the XML.

More methods are available for `IArchivedEntitySet`, `IArchivedEntity` and `IArchivedTypekey`. See the Gosu documentation for a full listing. Generate the Gosu documentation by navigating to the ClaimCenter bin directory and entering the following command:

```
gwcc regen-gosudoc
```

Incremental Upgrade

When ClaimCenter archives an entity, it records the current data model version on the entity. ClaimCenter upgrades an archived entity as the entity is restored. The upgrader executes the necessary archive upgrade triggers incrementally on each archived XML entity, according to the data model version of the archived entity.

An entity archived at one version might not be restored and upgraded until several intermediate data model upgrades have been performed. Therefore, do not delete your custom upgrade triggers.

Consider the following situation. Note that this example is for demonstration purposes only. The version numbers included do not represent actual ClaimCenter versions but are included to explain the incremental upgrade process. Each '+' after a version number indicates a custom data model change.

Guidewire data model changes

v6.0.0 – Entity does not have column X.

v6.0.1 – Guidewire adds column X to the entity with a default value of 100.

v6.0.2 – Guidewire updates the default value of column X to 200.

Implementation #1 upgrade path

v6.0.0+ – Start with version 6.0.0 data model with custom changes.

v6.0.0++ – More custom data model changes.

v6.0.0+++ – More custom data model changes.

v6.0.2+ – column X introduced with a default value of 200.

Result of upgrade of a claim archived at v6.0.0+: column X has value 200.

In this situation, version 6.0.1 was skipped in the upgrade path. Therefore, column X is added with the default value of 200 that it has in version 6.0.2.

Implementation #2 upgrade path

v6.0.0+ – Start with version 6.0.0 data model with custom changes.

v6.0.0++ – More custom data model changes.

v6.0.1+ – column X introduced with a default value of 100.

v6.0.2+ – column X already exists.

Result of upgrade of a claim archived at v6.0.0+: column X has value 100.

In this situation, column X is added in version 6.0.1 with the default value of 100. During the upgrade from version 6.0.1 to version 6.0.2, column X already exists. Therefore, the upgrader does not add the column. A custom trigger would be required to update the value of X from 100 to 200 during the upgrade from version 6.0.1 to 6.0.2.

Disabling the Scheduler

Before you start the server to upgrade the database, disable the scheduler for batch processes and work queues. Disabling the scheduler prevents batch processes and work queues from launching immediately after the database upgrade.

To disable the scheduler

1. Open the ClaimCenter 6.0.8 `config.xml` file in a text editor.
2. Set the `SchedulerEnabled` parameter to `false`.

```
<param name="SchedulerEnabled" value="false"/>
```
3. Save `config.xml`.

After you have successfully upgraded the database, you can enable the scheduler by setting `SchedulerEnabled` to `true`. This can be accomplished by performing the database upgrade using a WAR or EAR file that has the `SchedulerEnabled` parameter to `false`. After the upgrade is complete and verified, stop the server and deploy a new WAR or EAR file that differs from the first only by having `SchedulerEnabled` set to `true`. Finally, restart the server to activate the scheduler.

Suspending Message Destinations

Suspend all event message destinations before you upgrade the database to prevent ClaimCenter from sending messages until you have verified a successful database upgrade.

To suspend message destinations

1. Start the ClaimCenter server for the pre-upgrade version.
2. Log in to ClaimCenter with an account that has administrative privileges, such as the superuser account.
3. Click the **Administration** tab.
4. Click **Event Messages**.
5. Select the check box to the left of the **Destination** column to select all message destinations.

6. Click Suspend.

Resume messaging after you have verified a successful database upgrade.

Configuring the Database Upgrade

You can set parameters for the database upgrade in the ClaimCenter 6.0.8 `config.xml` file. The `<database>` block in `config.xml` contains parameters for database configuration, such as connection information. The `<database>` block contains an `<upgrade>` block that contains configuration information for the overall database upgrade. The `<upgrade>` block also contains a `<versiontriggers>` element for configuring general version trigger behavior and can contain `<versiontrigger>` elements to configure each version trigger.

This topic describes the parameters you can set for the database upgrade. For general database connection parameters, see “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*.

Configuring Column Removal on Oracle

The database upgrade removes some columns. For Oracle, you can configure whether the removed columns are dropped immediately or are marked as unused. Marking a column as unused is a faster operation than dropping the column immediately. However, because these columns are not physically dropped from the database, the space used by these columns is not released immediately to the table and index segments. You can drop the unused columns after the upgrade during off-peak hours to free the space. Or, you can configure the database upgrade to drop the columns immediately during the upgrade. By default, the ClaimCenter database upgrade marks columns as unused.

To configure the ClaimCenter upgrade to drop columns immediately during the upgrade, set the `oracleMarkColumnsUnused` attribute of the `<upgrade>` block to `false`. For example:

```
<database ...>
...
  <upgrade oracleMarkColumnsUnused="false">
    ...
  </upgrade>
</database>
```

Configuring Index Creation Parallelism on Oracle

You can configure the database upgrade to create indexes in parallel on Oracle databases. This can potentially reduce the time to upgrade the database if there are sufficient resources available on the database server. By default, indexes are not created in parallel on Oracle. Enable parallel index creation by setting the `createIndexInParallel` attribute of the `<upgrade>` block in the `<database>` block of `config.xml`. For example:

```
<database ...>
...
  <upgrade createIndexInParallel="2">
    ...
  </upgrade>
</database>
```

Set a numeric value for `createIndexInParallel` to specify the degree of parallelism for index creation during the upgrade. Or, set `value="default"` to have Oracle automatic parallel tuning determine the degree to use. Oracle determines the degree based on the number of CPUs and the value set for the Oracle parameter `PARALLEL_THREADS_PER_CPU`.

Enabling Collection of Tablespace Usage and Object Size

To enable collection of tablespace usage and object size data on Oracle, set the `collectstorageinstrumentation` attribute of the `<upgrade>` block to `true`. For example:

```
<database ...>
...
```

```

    <upgrade collectstorageinstrumentation="true">
    ...
  </upgrade>
</database>

```

A value of `true` enables ClaimCenter to collect tablespace usage and size of segments such as tables, indexes and LOBs (large object binaries) before and after the upgrade. The values can then be compared to find the utilization change caused by the upgrade.

Enabling Oracle Logging

You can enable logging of direct insert and create index operations during the database upgrade by setting `allowUnloggedOperations` to `false` in the `<upgrade>` block. For example:

```

<database ...>
...
  <upgrade allowUnloggedOperations="false">
  ...
  </upgrade>
</database>

```

The default is to run these statements with the `Nologging` option. Although Guidewire recommends that you backup the database before and after the upgrade, there could be reasons to log all operations. Some examples include Reporting, Disaster Recovery through Standby databases and Oracle Dataguard. To enable logging of direct insert and create index operations, set `allowUnloggedOperations` to `false`.

Storing Temporary Sort Results in tempdb

For SQL Server databases, you can specify to store temporary sort results in `tempdb` by setting the `sqlserverCreateIndexSortInTempDB` attribute of the `upgrade` block to `true`. By using `tempdb` for sort runs, disk input and output is typically faster, and the created indexes tend to be more contiguous. By default, `sqlserverCreateIndexSortInTempDB` is `false` and sort runs are stored in the destination filegroup.

If you set `sqlserverCreateIndexSortInTempDB` to `true`, you must have enough disk space available to `tempdb` for the sort runs, which for the clustered index include the data pages. You must also have sufficient free space in the destination filegroup to store the final index structure, because the new index is created before the old index is deleted. Refer to <http://msdn.microsoft.com/en-us/library/ms188281.aspx> for details on the requirements to use `tempdb` for sort results.

Adjusting Commit Size for Encryption

You can adjust the commit size for rows requiring encryption by setting the `encryptioncommitsize` attribute to an integer in the `<upgrade>` block. For example:

```

<database ...>
...
  <upgrade encryptioncommitsize="5000">
  ...
  </upgrade>
</database>

```

If ClaimCenter encryption is applied on one or more attributes, the ClaimCenter database upgrade commits batches of encrypted values. The upgrade commits `encryptioncommitsize` rows at a time in each batch. The default value of `encryptioncommitsize` varies based on the database type. For Oracle, the default is 10000. For SQL Server, the default is 100.

Specifying Filegroup to Store Sort Results for Clustered Indexes

For SQL Server databases, a version trigger recreates non-clustered backing indexes for primary keys as clustered indexes. This change improves the performance of claim archiving and claim purging operations.

Before recreating the indexes, the version trigger automatically drops (and later rebuilds) any referencing foreign keys and drops any clustered indexes on tables with a primary key.

If you are using filegroups, the upgrade recreates the clustered index in the OP filegroup. By default, the upgrade also stores the intermediate sort results that are used to build the index in the OP filegroup. You can configure the upgrade to instead use the tempdb filegroup for the intermediate sort results.

If you want the upgrade to store the intermediate sort results in the tempdb filegroup, set the `sqlserverCreateIndexSortInTempDB` attribute of the upgrade element to `true`.

```
<database ...>
...
<upgrade sqlserverCreateIndexSortInTempDB="true" />
...
</upgrade>
</database>
```

This option increases the amount of temporary disk space that is used to create an index. However, it might reduce the time that is required to create or rebuild an index when tempdb is on a different set of disks from that of the user database.

By default, `sqlserverCreateIndexSortInTempDB` is `false`.

Configuring Version Trigger Elements

The database upgrade executes a series of version triggers that make changes to the database to upgrade between versions. You can set some configuration options for version triggers in `config.xml`. Normally, the default settings are sufficient. Change these settings only while investigating a slow database upgrade.

The `<database>` element in `config.xml` contains an `<upgrade>` element to organize parameters related to database upgrades. Included in the `<upgrade>` element is a `<versiontriggers>` element, as shown below:

```
<database ...>
  <param ... />
  <upgrade>
    <versiontriggers dbmsperfinfothreshold="600" degreeofparallelismforinsertselects="2"/>
  </upgrade>
</database>
```

The `<versiontriggers>` element configures the instrumentation of version triggers. This element has two attributes: `dbmsperfinfothreshold` and `degreeofparallelismforinsertselects`.

The `dbmsperfinfothreshold` attribute specifies for each version trigger the threshold after which the database upgrader gathers performance information from the database. You specify `dbmsperfinfothreshold` in seconds, with a default of 600. If a version trigger takes longer than `dbmsperfinfothreshold` to execute, ClaimCenter:

- queries the underlying database management system (DBMS).
- builds a set of html pages with performance information for the interval in which the version trigger was executing.
- includes those html pages in the upgrader instrumentation for the version trigger.

You can completely turn off the collection of database snapshot instrumentation for version triggers by setting the `dbmsperfinfothreshold` to 0 in `config.xml`.

The `degreeofparallelismforinsertselects` attribute overrides a global default value (2) for the degree of parallelism for all tables involved in insert and select statements in version triggers that benefit from parallelism. This allows users to take advantage of their hardware when doing an upgrade. This attribute is only applicable for Oracle. Set `degreeofparallelismforinsertselects` to 1 if running on a single CPU machine.

The `<versiontriggers>` element can contain optional `<versiontrigger>` elements for each version trigger. Each `<versiontrigger>` element can contain the following attributes.

Attribute	Type	Description
<code>name</code>	String	The case-insensitive name of a version trigger.
<code>recordcounters</code>	Boolean	Controls whether the DBMS-specific counters are retrieved at the beginning and end of the use of the version trigger. Default is <code>false</code> . If <code>true</code> , then ClaimCenter retrieves the current state of the counters from the underlying DBMS at the beginning of execution of the version trigger. If the execution of the version trigger exceeds the <code>dbmsperfinfothreshold</code> , then ClaimCenter retrieves the current state of the counters at the end of the execution of the version trigger. ClaimCenter writes differences to the DBMS-specific instrumentation pages of the upgrade instrumentation.
<code>extendedquerytracingenabled</code>	Boolean	Oracle only. Controls whether or not to enable extended sql tracing (Oracle event 10046) for the SQL statements that are executed by the version trigger. Default is <code>false</code> . The output can be very useful when debugging certain types of performance problems. Trace files that are generated only exist on the database machine. They are not integrated into the upgrade instrumentation.
<code>queryoptimizertracingenabled</code>	Boolean	Oracle only. Controls whether or not to enable query optimizer tracing (Oracle event 10053) for the SQL statements that are executed by the version trigger. Default is <code>false</code> . The output can be very useful when debugging certain types of performance problems. Trace files that are generated only exist on the database machine. They are not integrated into the upgrade instrumentation.
<code>updatejoinhintbody</code>	String	Oracle only. Used to specify the body of an optimizer hint (without the surrounding <code>/*+ */</code>) for the upgrade of a join in the version trigger. If the update has a default override and the user specifies the empty string <code>""</code> as the value of the attribute, then no hint will be specified.

Starting the Server to Begin Automatic Database Upgrade

The database upgrade is an automatic process that occurs as you start the server with the upgraded configuration. The database upgrade normally completes in a few hours or less.

If the database upgrade stops before completing, then restore your database from the backup, correct any issues reported, and repeat the database upgrade.

IMPORTANT Before starting the upgrade, update database server software and operating systems as needed to meet the installation requirements of the target version. 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>.

WARNING Except for your first database upgrade trials, do not start the server until you have upgraded all rules. Otherwise, default validation rules execute. This could strand objects at a high validation level and make it impossible to edit parts of the object.

WARNING The database upgrade runs a series of version checks prior to making any changes. If any of these checks fail, the upgrade aborts and reports an error message. You can fix the issue, create an updated backup of the database and attempt the upgrade again without restoring from a backup. However, if you experience a failure during the version triggers or upgrade steps portion of the upgrade, refresh the database from a backup before attempting the upgrade again.

Test the Database Upgrade

Prior to attempting the database upgrade on a full-production database clone, test the database upgrade by doing the following:

1. Connected to the built-in H2 database, successfully start the built-in Quickstart (Jetty) application server with a merged configuration data model, including merged extensions, datatypes, fieldvalidators, and so forth.
2. Connected to an empty database on an Oracle or SQL Server database server, successfully start the Quickstart application server from the preceding step.
3. Connected to a restored backup of a production clone, start either the same Quickstart server from the preceding step or a third-party application server with your custom configuration.

Integrations and Starting the Server

Disable all integrations during the automatic database upgrade. Integration points might require updates due to changes in Guidewire APIs. See the *ClaimCenter New and Changed Guide* for specifics.

It is not necessary to have completely migrated integrations before attempting to start the server for the first time. If you have integrations that rely on non-Guidewire applications, do not expect these integrations to work the first time you start the server.

Understanding the Automatic Database Upgrade

As the database upgrade proceeds, it logs messages to the console as well as the log file describing its progress. The database upgrade process requires thousands of steps, divided into three phases. Due to the relational nature of a database, these phases must execute in a specific order for the upgrade to succeed.

During the first phase, the upgrader uses a set of *version triggers* to determine the first actions that need to take place. The database upgrader requires version triggers in order to perform the following types of tasks:

- changing a datatype (other than just length)
- migrating data from one table or column to another
- dropping a column

- dropping a table
- renaming a column
- renaming a table

Specific version triggers are described in this topic.

Many version triggers have version checks associated with them. These checks ensure that the database is ready for the associated version trigger. The database upgrade runs all checks before running any version triggers. If a check detects a problem, it reports the issue, including a sample SQL query to find specific problematic records. If a version check discovers an issue, the database upgrade stops before any version triggers are run. Therefore, it is not necessary to restore the database from a backup if a version check reports an error. Correct the issue and then create a new backup of the database. Then, if you encounter errors after the version check stage, you can restore a version of your database with the issue reported by the version check resolved.

In the second phase, the upgrader compares the target data model and the current database to determine how they differ. The upgrader makes changes to the database that do not require a version trigger during this phase.

Following this process, the third phase runs a subsequent set of version triggers. These triggers create actions that must be run last due to a dependency on an earlier phase.

After the database upgrade concludes, it reports issues that the upgrader encountered and did not complete.

You are responsible for correcting these issues. This might involve modifying the data model or altering the table manually. If you do not correct them, the next time you start the server you do *not* see a message that the database and the data model are out of sync. You must then use the `system_tools` command to verify the database schema.

Note: Given the complexity of database upgrade, Guidewire does not expose specific upgrade actions/steps to clients either in SQL or Java form. Any manual attempts to recreate or control the upgrade process can result in problems in the ClaimCenter database. Recovery from such attempts is not supported.

Version Trigger Descriptions

The database upgrade uses version triggers to perform the actions described by sections within this topic. If a version trigger has an associated version check, the check is described with the trigger. Review these descriptions to familiarize yourself with some of the changes and to understand version checks. If a version check reports an issue, review the error message and consult the description of the relevant version trigger for more information.

Updating the Database for Archiving Support

This step updates the database for the archiving feature. The upgrader performs the following actions:

- adds a `cc_claiminfo` table and populates it with data from `cc_claim`. The `cc_claiminfo` table stores basic claim information, the archive state of the claim, which database holds the archived claim, whether the claim is excluded from archiving, and reason for exclusion.
- renames the `ClaimID` column on `cc_claiminassoc` and `cc_bulkinvoiceitem` to `ClaimInfo`.
- creates and inserts data into `cc_bulkinvoiceiteminfo` based on data in `cc_bulkinvoiceitem`. Then, the upgrader drops the `ExposureID` and `ReserveLineWrapperID` columns from `cc_bulkinvoiceitem`.
- renames `cc_check.BulkInvoiceItemID` to `cc_check.BulkInvoiceItemInfoID`.
- drops the `Retired` column, if it exists, from `cc_contactinfo` and `cc_locationinfo`.
- creates an activity pattern called `Restore` for restoring archived claims from the archive.
- corrects the spelling of one `ArchiveState` typecode from `retriving` to `retrieving`.
- adds a `cc_periodpolicy.ClaimInfoID` foreign key column to `cc_claiminfo`.
- populates `cc_periodpolicy.ClaimInfoID` with the value of `cc_claim.ClaimInfoID` where `cc_claim.policyID` equals `cc_periodpolicy.policyID`.

Creating Region Zones for Regions

This trigger creates RegionZone entries for Regions. For each element in Region.Zones, the upgrader creates a RegionZone entry.

After the conversion, this trigger drops the cc_region.zones column.

Dropping and Recreating the Financials Report Upgrade Temp Table

This trigger drops and recreates the cc_financialsRptUpgrade table unless a forexadjustments column is present on the table. This table was used during upgrades from 2.1.x to 3.0.

Moving Document Template IDs

This step migrates document template IDs from Activity.Command to Activity.DocumentTemplate and ActivityPattern.DocumentTemplate. For each Activity.Command that begins with documentmerge.create(), the upgrader migrates the Activity.Command to Activity.DocumentTemplate and ActivityPattern.DocumentTemplate. For matching records, the upgrader sets Activity.Command to null.

Populate Holiday List from Configuration

This step populates the cc_holiday and cc_holidaytag tables with Holiday records, creating a Holiday for each date set to the HolidayList parameter in config.xml.

Checking Role Privilege Public IDs

This step checks if any rows in cc_roleprivilege contain a PublicID with an integer portion greater than the next ID for the table from the key generator. If any rows match this condition, this version check reports an error message. The error message includes the offending ID, PublicID pairs and an SQL query to find the rows. This issue can be resolved two ways:

- Update the PublicIDs so that the integer portion is less than the next available ID.
- Or, change the prefix of the PublicID fields so that it does not match the application prefix.

Upgrading ABContact Permissions

This step performs the following upgrades to ABContact permissions:

- Grant abviewsearch permission to all roles that already have abview permission.
- Grant abdelete permission to all roles that already have abcreate.
- Grant abdeletepref permission to all roles that already have abcreatepref.
- Grant abedit permission to all roles that already have abeditpref.

Deleting TAccounts that Lack Owning Reserve Line

This step deletes any TAccounts that do not have an owning ReserveLine. This could happen if a ReserveLine was created but then removed from the bundle prior to commit, allowing the TAccounts to be committed without the ReserveLine.

Upgrading TAccounts and ReserveLines

This step upgrades TAccounts. This version trigger performs the following steps:

- Creates the TmpTAccountReserveLine table and populates it.
- Adds the ReserveLineId, DebitRptBalance, CreditRptBalance and NumContributingTxns columns to the TAccount table.
- Populates these four new columns.

- Changes new columns to be non-nullable.
- Drops the TAccount foreign keys from the ReserveLine table.

Adding Gross Claim Amount and Currency Columns to Check Reports

This step adds and populates GrossClaimAmount and Currency columns on CheckRpt. The upgrader sets GrossClaimAmount to the current GrossAmount of the CheckRpt record. The upgrader sets Currency to the DefaultApplicationCurrency defined for the system in config.xml. By default, this is usd.

Upgrading the Policy Model

This step first checks for the following conditions:

- Any third-party properties that have an employment code. A third-party property is a property that is not part of a policy.
- All extensions created on PolicyVehicle or PolicyProperty are also created and have the same type on RiskUnit.
- Any non-retired Policy entities with no Claim associated with them.

If any of these checks fails, the upgrader reports an issue, provides a SQL query to find the problem and aborts. Resolve any issues found before running the database upgrade again.

If the version checks succeed, the upgrader performs the following actions:

- renames EmploymentClass to ClassCode and EmploymentData.EmpClassID to EmploymentData.ClassCodeID. If you have extensions defined on EmploymentClass, see “Employment Class” on page 245 for information on keeping the extension data.
- renames PhysicalProperty to PolicyLocation. This renames the main table cc_property, the staging table ccst_property, and the shadow table cct_property to cc_policylocation, ccst_policylocation and cct_policylocation, respectively. If you have extensions defined on PhysicalProperty, see “Physical Property” on page 245 for information on keeping the extension data.
- renames PolicyLocation.Location to PolicyLocation.LocationNumber.
- adds and populates a PolicyID column on PolicyLocation. The PolicyID is set to the PolicyID of the corresponding PolicyProperty record.
- creates a Building table, cc_building, and populates it with a record for each PhysicalProperty with a BuildingNumber.
- creates a RiskUnit table, cc_riskunit. It then moves data from PolicyVehicle and PolicyProperty to cc_riskunit, updates coverages to be an array on RiskUnit instead of Vehicle or Property, and drops the PolicyVehicle and PolicyProperty tables.
- drops the ClassCode.PropertyID (formerly EmploymentClass.PropertyID) and PolicyLocation.BuildingNumber (formerly PhysicalProperty.BuildingNumber) columns.
- adds a ClassCode.PolicyID column and populates it.

There are other changes to the policy model that are not handled by the database upgrader. For a full description of policy model changes, see “Policy Data Model Changes” on page 248.

Dropping Unused Contact Columns

This step removes the following unused columns from Contact:

- AddressBookVersion
- InAddressBoook
- ReservedID1 - ReservedID5
- SyncedVersion

Creating TAccounts for Rejection

This step performs the following TAccount changes:

- Creates two new TAccounts: `CommittedErodingPayments` and `Recoveries`.
- Points all `TAccountLineItems` that are currently debiting the `Reserves` TAccount to instead debit the new `CommittedErodingPayments` TAccount.
- Points all `TAccountLineItems` that are currently crediting the `RecoveryReserves` TAccount to instead credit the new `Recoveries` TAccount.
- Copies the debit balance from the `Reserves` TAccount to be the debit balance of the new `CommittedErodingPayments` TAccount.
- Copies the credit balance from the `RecoveryReserves` TAccount to be the credit balance of the new `Recoveries` TAccount.
- Zeroes out the debit balance on the `Reserves` TAccount and the credit balance on the `RecoveryReserves` TAccount.

Populating Bulk Invoice Currency

This step adds and populates the following columns to `BulkInvoice`:

- `TotalReportingAmount`
- `TotalTransactionAmount`
- `ApprovedReportingAmount`
- `ApprovedTransaction amount`

For a `BulkInvoice` `bi`:

- `TotalReportingAmount = TotalTransactionAmount = sum of (bi.InvoiceItem.Amount)`
- `ApprovedReportingAmount = ApprovedTransactionAmount = sum of (bi.InvoiceItem.Amount where InvoiceItem.Status is APPROVED, SUBMITTING, or SUBMITTED).`

Copying TAccount Line Item Amounts

This step creates a `RptAmount` column on `TAccountLineItem` and copies the existing `Amount` column for each `TAccountLineItem` to `RptAmount`.

Dropping Administrative Tables

This step drops the following tables:

- `cctt_usersearchcriteria`
- `cctt_roleconstraint`
- `cc_tmppagglimitupd`

The upgrader then creates a new `cc_tmppagglimitupd` table.

Upgrading Note and Document Permissions

This step adds the following system permissions:

- `docedit`
- `docview`
- `noteedit`
- `noteview`

Adding Activity Patterns

This step adds the `ActivityPatterns` `CheckDenied` and `RecoveryDenied`, for denial of checks and recoveries. ClaimCenter creates these activities when a check or recovery is denied by a downstream system. See “Downstream Denials of Recoveries and Checks” on page 160 in the *Application Guide*.

Dropping Column from Risk Unit

This trigger drops the `cc_riskunit.WCBuildingID` column.

Creating Injury Incidents

This step first performs a series of checks on the database.

For the following typelists, the upgrader checks that each row in the workers' compensation version of the type-list matches a typekey in the upgraded and merged typelist configuration file. The matching typekey must have the same typecode and name attributes.

- `BodyPartType`
- `DetailedBodyPartType`
- `DetailedInjuryType`
- `InjuryType`
- `MedicalTreatmentType`

For non-injury exposures, the upgrader checks that all of the injury-related columns are null, including:

- `LeadingInjury`
- `DetailedInjury`
- `PrimaryBodyPart`
- `DetailedBodyPart`
- `MedicalTreatment`
- `WCLeadingInjury`
- `WCDetailedInjury`
- `WCPrimaryBodyPart`
- `WCDetailedBodyPart`
- `WCMedicalTreatment`
- `MPLLeadingInjury`
- `MPDetailedInjury`
- `MPPPrimaryBodyPart`
- `MPDetailedBodyPart`
- `MPMedicalTreatment`
- `Impairment`
- `LWImpairment`
- `LostWages`

For each Exposure, the upgrader checks that the `DetailedInjury`, `LeadingInjury` and `MedicalTreatment` columns on an Exposure have compatible values with the related WC- and MP- columns.

- If the WC- column is not null, the other two columns must be null.
- If the main column is populated, the MP- version must be populated with the same value or null. For example, if `DetailedInjury` is populated, `MPDetailedInjury` must have the same value or be null.
- If the MP- column is populated, the main version must be populated with the same value or null. For example, if `MPDetailedInjury` is populated, `DetailedInjury` must have the same value or be null.

The upgrader also checks that:

- only one Exposure is linked to each injury-related Incident. The upgrader will be converting the injury-related incidents to `InjuryIncidents`.
- Exposures pointing to the same Incident have the same `Severity` value. The upgrader will later move the `Severity` column to Incident.
- for each Claim, there is no more than one injury description in `ClaimText`.
- only one out of two impairment fields (`Impairment` and `LWImpairment`) are set on each Exposure.
- Any additional columns on `BodyPartDetails` are defined as extensions. The upgrader will move the `cc_bodypart` table from being an array on Claim to being an array on `InjuryIncident`. In the process of this

move rename the existing table and recreate the `cc_bodypart` table from the metadata. If the metadata does not contain all the same columns as the existing table, then the upgrade could result in a loss of data. Any columns not defined in the metadata will be renamed but will not be copied back into the recreated table. This version check ensures that the metadata is not missing any columns. If it finds any it stops the upgrade and gives an error message asking the user to either add in the missing columns or delete them before the upgrade starts.

In ClaimCenter 5.0 some individual address fields which used to be on `MobilePropertyIncident` are moved into a separate `Address` object. The fields are:

- `LocationCity`
- `LocationState`
- `LocationStreet`
- `LocationZip`

In 5.0 these are replaced by a single `LocationAddress` field. In 4.0 these fields were on `MobilePropertyIncident`, but specified in extensions so you could have moved them to a different `Incident` subtype. The `LocationAddress` field is also movable.

The upgrader checks that if any of the location fields are set on an `Incident`, then the corresponding incident type has the `LocationAddress` foreign key field. If it does not, the upgrader aborts. This trigger requires that the `LocationAddress` foreign key extension field is visible to the `MobilePropertyIncident`.

If all these checks succeed, the upgrader creates `InjuryIncident` entities. The upgrader moves injury-related information from `Claim` and `Exposure` to `InjuryIncident`. For details on these changes, see “Injury Information Moved to New `InjuryIncident` Entity” on page 250.

Refactoring Assignments

The upgrader renames the user and group columns on `UserRoleAssignment` to `assigneduser` and `assignedgroup` to match the `Assignable` interface. The upgrader also drops the `Assignment` and `AssignmentRole` tables.

Dropping the OrgOrg Entity

This step drops the `cc_orgorg` and `cct_orgorg` tables.

Dropping Unused Column from User

This step first checks if a `CarrierInternalUser` column is defined in the extensions for the `User` entity. If an extension is defined, the upgrader does nothing and proceeds to the next step.

If a `CarrierInternalUser` extension is not defined, the upgrader checks for any `User` records with `CarrierInternalUser` equals 0 (false). If the upgrader finds any such `User` records, it reports an error and aborts. Either change the value of `CarrierInternalUser` to 1 (true) for all `User` records, drop the column, or create an extension for it before running the database upgrade again.

If no `CarrierInternalUser` extension is defined on `User`, and `User.CarrierInternalUser` is not set to 0 for any `User` records, the upgrader drops the `CarrierInternalUser` column.

Upgrading Workflows

The upgrader performs a series of steps to upgrade workflows. The upgrader:

- sets the `Subtype` column for all `Workflow` entities to `MetroReportWorkflow`.
- drops the `AbstractWorkflow` typelist table `cctt_abstractworkflow`.
- drops the `WorkflowSearchCriteria` typelist table `cctt_workflowsearchcriteria`.
- drops the `Type` column from `Workflow`.
- converts the `WorkflowLog` to a versionable entity.

Dropping Column from Check Staging Table

This step drops the `BulkInvoiceItemInfoID` column from the `ccst_check` staging table. This column was not intended to be loadable.

Dropping Lookup Table Name Typelist

This step drops the lookup table name typelist table `cctl_lookuptablename`.

Converting Security Mappings

This step converts records in `security-mapping.xml` into `ReportGroup` records in the database. The upgrader converts each default mapping and custom-defined mapping. The `ReportGroup` entity is introduced in ClaimCenter 5.0.

Moving Inactive Messages to History Table

This step moves all inactive messages to the `MessageHistory` table. The `MessageHistory` table is introduced in ClaimCenter 5.0. The upgrader removes the inactive messages from the `Message` table after it has copied them successfully to `MessageHistory`.

If you have defined extensions to the `Message` entity, define the same extensions on `MessageHistory`.

Updating Messaging

This trigger drops the `cc_message.SendOrder` column.

The trigger adds a `MessageTime` column to `cc_message` and initializes this column to the current time.

Updating Role Privilege Permission Typecodes

`RolePrivilege.Permission` can no longer equal `rptgrpadmview` or `rptgrpadmmanage`. This upgrade trigger reassigns these typecodes to `reporting_view` and `reporting_admin`, respectively.

Dropping Aggregate Limit Report Staging Table

Aggregate limit reports can no longer be loaded. This step removes the aggregate limit report staging table, `ccst_agglimitrpt`.

Dropping Display Setting Typekey and References

This trigger drops the `DisplaySettingType` type key. The trigger drops the `cctl_displaysettingtype` and `cctt_displaysettingtype` tables and the `cc_user.DisplaySetting` column.

Dropping State Column from ClaimInfo Table

This trigger drops the `cc_claiminfo.State` column.

Dropping and Recreating Financials Calculation Temp Tables

This trigger drops and recreates the `cc_TmpCheckRpt` and `cc_TmpCheckRptCheckGroup` tables.

Dropping Revision Distributions Collected Information from Data Distribution Table

This trigger drops the `cc_databasedatadist.RevisionDistsCollected` column. ClaimCenter no longer uses revisioned entities.

Dropping Obsolete Archiving Tables

This trigger drops the `cc_archivemaxkey` and `cct_archivemaxkey` tables.

Dropping Obsolete Temporary Tables

This trigger drops the following obsolete temporary tables that were used for upgrades from ClaimCenter 2.1.x to 3.0.x:

- `cc_tempccmap`
- `cct_tempccmap`
- `cc_tempabupgrade`
- `cct_tempabupgrade`

Dropping ID Offset from New TAccount Join Temporary Table

This step drops the `cc_tmpnewtacctjointbl.idoffset` column. This column was introduced in ClaimCenter 5.0.

Initializing Official ID Subtypes

The `OfficialID` entity was previously final. It now requires a subtype. This trigger adds and initializes a new subtype column on `cc_officialID`. For each row in `cc_officialID`, the trigger sets the subtype column to the base typecode `OfficialID`.

Dropping Unused Sample Tables

This trigger drops the `cctt_samplea` and `cctt_sampleb` tables.

Dropping Retired Column from Contact Information and Location Information

This trigger drops the `Retired` column from the `cc_contactinfo` and `cc_locationinfo` tables.

Dropping Process History Columns

This step removes the following columns from `cc_ProcessHistory`:

- `PhaseInProgress`
- `PhaseNumber`
- `PhasesToExecute`

Renaming Message Sink Columns

This step renames the `MessageSink` column on the `cc_Message` and `cc_MessageHistory` tables to `DestinationID`.

Dropping Message Sink Tables

This step drops the `cc_loadmsgsinkinsertselect` table and its shadow table `cct_loadmsgsinkinsertselect`.

Creating Subtypes for WorkflowLogEntry

ClaimCenter 6.0 adds three subtypes of the `WorkflowLogEntry` entity: `WorkflowTextLog`, `WorkflowActionLog`, and `WorkflowUserLog`.

This step adds a `cc_workflowlog.subtype` column and sets the value for all rows to the `WorkflowTextLog` subtype.

Upgrading Data Distribution Report Model

This step first removes data distribution reports from the database.

Then, it drops the following tables:

- `cc_revisiondatadist`
- `cc_revisionsizecntdd`
- `cct_revisiondatadist`
- `cct_revisionsizecntdd`

The upgrader then drops the `cc_databasedatadist.RevisionDistsCollected` column, if it exists.

Finally, the upgrader converts the datatype of the following `cc_clobcoldatadist` columns to decimal with a precision of 19 and scale of 0:

- `AverageLength`
- `MaximumLength`
- `MinimumLength`

Dropping Platform Region Type Column

This step removes the `cc_region.regiontype` column. Regions in ClaimCenter 6.0.8 are linked to zones through the new `RegionZone` entity.

Upgrading Holiday Zone Codes

As businesses expand into new geographical areas, there is an increasing chance that a holiday zone code is duplicated. For example, the holiday zone code CA could be used to represent California or Canada. To address this issue, ClaimCenter adds `ZoneType` and `Country` properties to `HolidayZone`, enabling you to more precisely define the holiday zone.

The database upgrader first checks that all holiday zone code records have a `cc_holiday_zone.code` value that exists in `cc_zone.code`. If any records are found where `cc_holiday_zone.code` does not exist in `cc_zone.code`, the upgrader logs a message. This message includes a query to find the violating rows. Use this query to identify the rows and then update the rows to use a zone code that does exist in `cc_zone`.

This step then adds and populates two non-nullable typekey columns to `cc_holiday_zone`: `ZoneType` and `Country`. In cases in which `HolidayZone.Code` matches a single `Zone` record, `HolidayZone` takes its `ZoneType` and `Country` values from the `Zone`. In cases in which `HolidayZone.Code` matches multiple `Zone` records, `Holiday.ZoneType` and `Holiday.Country` are populated with the typecode `unknown`. These `HolidayZone` records could not be conclusively populated with `ZoneType` and `Country` values from corresponding `Zone` records. Manually edit these unknown values to real-world values. For example, you could have two `Zone` records with Code CA, representing California and Canada. After the upgrade, these `HolidayZone` records have a `ZoneType` and `Country` value of `unknown`. Manually edit these records to set the `ZoneType` and `Country` typekeys.

For more information, see “Columns Added to HolidayZone Entity” on page 41 in the *New and Changed Guide*.

Adding Hash of Full Path to SREE Reports

This step adds a `HashOfFullPath` column to `SREEReport` and populates it by calculating a sha1 hash of `SREEReport.FullPath` for each row. ClaimCenter 6.0 calculates the `HashOfFullPath` value each time the `SREEReport.FullPath` column is set. This step removes the uniqueness index from `FullPath` and adds a uniqueness index to `HashOfFullPath`.

Checking Activity Pattern Code Uniqueness

This step checks that no `ActivityPattern` records have the same `Code`. If the upgrader finds duplicate activity pattern codes, it writes an error message to the log. This message includes a query to find the duplicate activity codes. Update any duplicate activity pattern codes to unique values before proceeding.

Dropping ExtractReady from Organization Zone Admin

This step removes the ExtractReady column from cc_organizationzoneadmin.

Expanding Geocoding Precision

This step expands the precision defined for the cc_address latitude and longitude decimal columns.

Dropping Columns from Contact Category Score

This step drops the Admin and ExtractReady columns from cc_ContactCategoryScore.

Migrating LodgingProvider

Prior versions included extension columns on PropertyIncident called LodgingNights and LodgingRate. In ClaimCenter 6.0, Guidewire eliminated those columns and added a LodgingProvider array. This step creates cc_LodgingProvider rows where there was data in the two removed columns. If the LodgingProvider entity does not contain LodgingNights and LodgingRate columns, the trigger does nothing. The trigger drops the source columns from PropertyIncident if they are no longer defined as extensions and the trigger successfully created rows in cc_LodgingProvider.

LodgingProvider.LodgingNights exists only for upgrade. In ClaimCenter 6.0 the number of nights is determined using LodgingProvider.StartDate and LodgingProvider.EndDate. If either of those dates is null, ClaimCenter checks LodgingProvider.LodgingNights for data from the upgrade and displays it if present. ClaimCenter never writes to the LodgingProvider.LodgingNights field after upgrade.

Removing instrumentedbatchjobid Column from Work Item Tables

This step removes the instrumentedbatchjobid column from the following work item tables:

- ClaimValidationWorkItem
- ContactAutoSyncWorkItem
- GeocodeWorkItem
- ReviewSyncWorkItem
- WorkflowWorkItem

Dropping Profiler Tables

This step removes the following tables related to the profiler:

- cc_dbProfilerTagPath
- cct_dbProfilerTagPath
- cc_profilerData
- cct_profilerData
- cc_ProfilerPathEdge
- cct_ProfilerPathEdge
- cc_profilerPeriod
- cct_profilerPeriod

Dropping Temporary Contact Tables

This step removes the following tables:

- cc_tempaddressmap
- cc_tempcontactcontact
- cc_tempcontactmap
- cct_tempaddressmap
- cct_tempcontactcontact
- cct_tempcontactmap

Dropping Staging Table Region Type

This step removes the `ccst_region.regiontype` column. Regions in ClaimCenter 6.0.8 are linked to zones through the new `RegionZone` entity.

Dropping Instrumentation Tables

ClaimCenter 6.0 stores batch process performance information in the profiler rather than in instrumentation tables. This step removes the following tables:

- `cc_instrumentedjob`
- `cct_instrumentedjob`
- `cctt_instrumentedjob`
- `cc_instrumentedjobappserver`
- `cct_instrumentedjobappserver`
- `cc_instrumentedjobparam`
- `cct_instrumentedjobparam`
- `cc_instrumentedjobtask`
- `cct_instrumentedjobtask`
- `cctt_instrumentedjobtask`
- `cc_instrumentedjobthread`
- `cct_instrumentedjobthread`

Upgrading Typelist Tables

This step first drops all staging tables. Then for each typelist table, this step expands the size of the typecode column from 22 to 50.

Adding Claim and Policy Currency Columns

This step adds a Currency column to `cc_claim` and `cc_policy` and populates this column with the reporting currency. The reporting currency is the value set to `DefaultApplicationCurrency` in `config.xml`.

See “Policy Currency” on page 26 in the *New and Changed Guide*.

Checking for Negative Values in Non-Negative Incident Columns

A number of columns that were integer datatypes have been updated to only store non-negative integers. This version check reports an error if any of the non-negative columns for Fixed Property, Property or Vehicle incidents contains a negative value. The columns that this version check inspects are listed by incident type:

Fixed Property

- `NumSprinkler`
- `NumSprinkOper`
- `NumStories`

Property

- `MealsDays`
- `MealsPeople`
- `MealsRate`

Vehicle

- `OdomRead`

Removing Reinsurance Status Typekey and References

This step removes the `ReinsuranceStatus` typekey and references to that typekey. In ClaimCenter 6.0, the reinsurance status feature is replaced by support for reinsurance thresholds. You can preserve reinsurance status data by creating a `ReinsuranceStatus` extension on `Claim` and a `ReinsuranceStatus` typekey. This process is described in “Reinsurance Status” on page 130.

The upgrader first checks if there is data in `cc_claim.ReinsuranceStatus`. If there is data in that column, and no `ReinsuranceStatus` typekey extension defined on `Claim`, the upgrader reports a problem. If you do not want the data, drop the data in `cc_claim.reinsurancestatus` and run the upgrader again. If you do want to keep the data, create a `ReinsuranceStatus` typekey extension on `Claim` and a `ReinsuranceStatus` typelist. If the upgrader detects this extension in the data model, it retains data in `cc_claim.reinsurancestatus` and does not drop the `cctl_reinsurancestatus` and `cctt_reinsurancestatus` tables.

If there is no data in `cc_claim.ReinsuranceStatus`, and there is no `Claim.ReinsuranceStatus` column extension, the upgrader drops the `cc_claim.ReinsuranceStatus` column and the `cctl_reinsurancestatus` and `cctt_reinsurancestatus` tables.

Adding Activity Pattern for Reinsurance Review

This step adds the Claim Reinsurance Review activity pattern.

Adding Claimant Flag on ClaimContact

This step adds a `ClaimantFlag` boolean column to `cc_claimcontact`, if `cc_claimcontact.ClaimantFlag` does not already exist. The upgrader then populates the new column.

For records in `cc_claimcontact` where `cc_claimcontact.ID` equals `cc_claimcontactrole.ClaimContact` and `cc_claimcontactrole.Role` equals `claimant`, and `cc_claimcontactrole.Retired` equals 0, the upgrader sets `cc_claimcontact.ClaimantFlag` to 1. Otherwise, the upgrader sets `cc_claimcontact.ClaimantFlag` to 0.

Adding Transaction and Reporting Amount to Deduction

This step first renames `cc_deduction.Amount` to `cc_deduction.ClaimAmount`.

This step then adds a `cc_deduction.TransactionAmount` column, if the column does not already exist. This column was added in ClaimCenter 5.0.

The upgrader then populates `cc_deduction.TransactionAmount` with the value of `cc_deduction.ClaimAmount`. The upgrader only populates `cc_deduction.TransactionAmount` if it is null.

This step also adds a `cc_deduction.ReportingAmount` column and populates this new column with the value of `cc_deduction.ClaimAmount`.

Adding FixedClaimAmount and FixedReportingAmount to CheckPortion

This step first renames `cc_checkportion.FixedAmount` to `cc_checkportion.FixedTransactionAmount`.

The upgrader then adds a `cc_checkportion.FixedClaimAmount` column, if the column does not already exist. This column was added in ClaimCenter 5.0.

The upgrader then populates `cc_checkportion.FixedClaimAmount` with the value of `cc_checkportion.FixedTransactionAmount`. The upgrader only populates `cc_checkportion.FixedClaimAmount` if it is null.

The upgrader then adds `cc_checkportion.FixedReportingAmount` and populates this new column with the value of `cc_checkportion.FixedClaimAmount`.

Removing Unused Deductible Columns

ClaimCenter 6.0 changes the data model for deductibles. ClaimCenter includes a new `Deductible` entity. Each deductible is associated with a coverage, and by default gets its initial amount from the `Coverage.Deductible` field. A coverage can be associated with zero or one deductible. The `TransactionLineItem` entity includes a foreign key to `Deductible`. The `Deductible` entity is not loadable through staging tables. For details on the deductible data model, see “Configuring Deductibles” on page 633 in the *Configuration Guide*. See also “Deductible Handling” on page 183 in the *Application Guide*.

This step checks that the following columns are empty and not defined as extensions in the data model and then removes them:

- `cc_claim.DeductibleStatus`
- `cc_transaction.DeductibleAmount`
- `cc_transaction.DeductiblePaid`
- `cc_transaction.DeductibleSubtracted`
- `cc_transaction.NetAmount`
- `cc_transaction.OriginalAmount`

If any one of these columns contains data, and is not defined as an extension, the upgrader reports an error, does not remove the data and aborts the upgrade.

If you want to preserve data in any of these columns, define the column as an extension in the data model. If a column is defined as an extension, the upgrader does not remove the column or its data. See “Deductible Fields” on page 129.

Renaming ISO Match Report

This step renames the ISO match report table and accompanying staging and shadow tables, as follows:

Old name	New name
<code>cc_isomatchreport</code>	<code>cc_ExposureISOMatchReport</code>
<code>ccst_isomatchreport</code>	<code>ccst_ExposureISOMatchReport</code>
<code>cct_isomatchreport</code>	<code>cct_ExposureISOMatchReport</code>

Remapping Incidents

This step first checks that:

- for any incident with multiple exposures, the upgraded exposure types agree on the incident subtype.
- for each required incident type change, the new incident type has the same set or a superset of the fields of the old incident.
- for each required incident type change, the new incident type has the same set or a superset of the arrays of the old incident.
- for each required incident type change, any references to the incident are still valid with the new type.

This step then updates `cc_exposure.ExposureType` to match `cc_exposure.CoverageSubtype` if the `CoverageSubtype` to `ExposureType` map has changed.

The upgrader then updates Incident subtypes to match the `ExposureTypes` of referring Exposures.

ClaimCenter 6.0 adds new exposure and incident types for the homeowners policy types. These exposure and incident types are described in the following table, along with their old counterparts for homeowners coverage subtypes.

New Exposure Type	New Incident Type	Old Exposure Type	Old Incident Type	Coverage Subtypes
Dwelling	Dwelling Incident	Property	Fixed Property Incident	<ul style="list-style-type: none"> Dwelling - Property-Damage Earthquake - Dwelling - PropertyDamage Flood - Dwelling - PropertyDamage Mold - Dwelling - PropertyDamage
Living Expenses	Living Expenses Incident	Loss of Use	Property Incident	<ul style="list-style-type: none"> Loss of Use Damage
Other Structure	Other Structure Incident	Property	Fixed Property Incident	<ul style="list-style-type: none"> Other Structure - Property Damage Earthquake - Other Structure - Property-Damage Flood - Other Structure - Property Damage
Content	Property Content Incident	Personal Property	Mobile Property Incident	<ul style="list-style-type: none"> Personal Property Damage Scheduled Property - Personal Property Damage Earthquake - Personal Property Damage Flood Personal Property - Personal Property Damage

The new incidents are subtypes of FixedPropertyIncident.

If you do not want to use any of the new relationships, you can change your CoverageSubtype to ExposureType map using Studio 6.0 before you run the database upgrade.

You can export line of business relationship information from Studio 5.0 and 6.0.

To export line of business relationship information

1. From the command line navigate to ClaimCenter\bin.
2. Enter `gwcc studio` to launch Studio.
3. In the Studio **Resources** pane, click **configuration** → **Lines of Business** → **PolicyType**.
4. In the Navigation panel, right-click **Homeowners** and select **Export Branch** → **HTML** or **Export Branch** → **CSV**.
5. Select a location for the exported HTML or CSV file.
6. Click **Save**.

Adding Activity Patterns for Homeowner Exposures

This step adds the Damaged Items and Living Expenses activity patterns used for the homeowners line of business.

Converting Catastrophe States to Catastrophe Zones

ClaimCenter 6.0 changed from using `CatastropheStates` to `CatastropheZones`. `CatastropheStates` were introduced in ClaimCenter 5.0, so this step only applies if upgrading from ClaimCenter 5.0.x.

You can choose to continue to use `CatastropheState` instead of the `CatastropheZone` functionality. To do so, you create a `CatastropheState` extension table `ccx_catastrophestate`. If the upgrader detects this table in the data model, it renames `cc_catastrophestate` to `ccx_catastrophestate`, preserving data in the table. In this case, this upgrade step stops after the rename operation.

For instructions to preserve `CatastropheState`, see “Retaining Catastrophe States Instead of Catastrophe Zones” on page 127.

If there is no `ccx_catastrophestate` extension, this upgrade step checks that each `cc_catastrophestate.State` code exists as a zone code in `cc_zone` with zone type of state or province. If this check fails, the upgrader logs an error message, including a query to find the catastrophe states that can not be upgraded.

If each `cc_catastrophestate.State` code exists as a zone code in `cc_zone`, the upgrader converts each `CatastropheState` to its `CatastropheZone` equivalent. The upgrader then deletes the `cc_catastrophestate` table and its’ staging and shadow tables, `ccst_catastrophestate` and `cct_catastrophestate`.

The ClaimCenter 5.0.x version of `CatastropheState` included a `Comments` column. If there is any data in this column, and there is a `Comments` column on `CatastropheZone`, the database upgrader copies the data to the `CatastropheZone.Comments` column. The `CatastropheZone` entity does not have a `Comments` column by default. If you have comments in `CatastropheState`, and you want those comments moved to the new `CatastropheZone` entity, create a `Comments` column extension on `CatastropheZone`. Or, if you do not want to preserve the comments from `CatastropheState`, you can alter the `cc_catastrophestate` table to remove the `Comments` column before beginning the upgrade. If the upgrader detects data in the `CatastropheState.Comments` column, and does not detect a `Comments` column extension on `CatastropheZone`, it reports an error message and aborts the upgrade.

For instructions, see “Preserving Comments from Catastrophe State” on page 128.

Adding Currency Column to Authority Limit Profiles

This step adds a `Currency` column to `cc_authorityprofile` and populates the new column with the reporting currency. The reporting currency is the value set to `DefaultApplicationCurrency` in `config.xml`.

See “Currency Aware Authority Limit Profile” on page 30 in the *New and Changed Guide*.

Dropping Source File from Question Set Tables

This steps drops the `SourceFile` column from the following tables:

- `cc_question`
- `cc_questionchoice`
- `cc_questionfilter`
- `cc_questionset`
- `cc_questionsetfilter`

Dropping extractready Column from Certain Tables

This step removes the `extractready` column from the following tables:

- `cc_bulkinvoiceitem`
- `cc_claimassociation`
- `cc_claiminassociation`
- `cc_claiminfo`
- `cc_claiminfoaccess`

- cc_contactinfo
- cc_exposureismatchreport
- cc_isomatchreport
- cc_locationinfo
- cc_message
- cc_periodpolicy
- cc_tmplclaimaccess
- cc_tmpstclaimaccess
- cc_tmptsteclaimaccess
- cc_wcbenefitfactordetail

Checking Payment Type

This version check looks for Payments with a null PaymentType. If the version check finds Payments with a null PaymentType, it reports an error. The error message includes a SQL query to identify the Payments with a null PaymentType. You must set a PaymentType for each Payment that this version check reports.

Removing Availability Script from Question Set

This trigger removes the cc_questionset.availabilityscript column. This column was not used by ClaimCenter.

Removing Example Cost Category and Extension Array Entities

This step first checks for any data in the ccx_costcats and ccx_extarray tables. These tables are examples and are not supported for upgrade. If the check finds any records in one of these tables, then it reports an error. If the check passes, this step runs a version trigger to drop the following tables:

- cct_costcats
- ccst_costcats
- ccx_costcats
- cct_extarray
- ccst_extarray
- ccx_extarray

Checking Claim Stubs for Removed Archive States

This version check looks for ClaimInfo records with an ArchiveState of either archiving or markedforarchive. If the version check finds any such records, then it aborts the upgrade and prints a warning message. The database cannot be upgraded when there are items being archived.

Ensuring Claim Loss Locations Are Unique

This version check looks for loss location addresses shared by multiple claims. This version check reports an error if it detects loss location addresses shared by multiple claims. The error message includes an SQL query to identify these claims. If you want multiple claims to share a loss location, update each claim to refer to its own copy of the shared address.

Checking History Records

This version check looks for History rows that have a HistoryType of markedforarchive. The markedforarchive HistoryType is removed in ClaimCenter 6.0. If this version check finds any History rows that have a HistoryType of markedforarchive, it reports an error and provides an SQL query to find the rows. You must manually remove these rows or add the markedforarchive typecode back to perform the upgrade.

Checking for Claims That Were Archived Incorrectly

This version check looks for the following issues with archived claims:

- transferred check exists in the archive database
- Claim referenced by a BulkInvoiceItem
- primary database contains BulkInvoiceItemInfo of an archived Claim.

This version check reports an error if it discovers any of these conditions. Contact Guidewire Support for assistance fixing any issues reported by this version check.

Adding Claim Column to BulkInvoiceItemInfo

This version trigger adds a ClaimID column to cc_biiteminfo. The trigger populates this column based on the ClaimID of the ClaimInfo for the BulkInvoiceItem.

Adding Reporting Columns to Claim and Exposure Report Tables

This version trigger adds and populates the following columns on cc_claimrpt and cc_exposurert:

- AvailableReservesReporting
- FuturePaymentsReporting
- OpenRecoveryReservesReporting
- OpenReservesReporting
- RemainingReservesReporting
- TotalPaymentsReporting
- TotalRecoveriesReporting

The trigger populates the columns with the values contained in the non-reporting versions of these columns. For example, the trigger populates cc_claimrpt.AvailableReservesReporting with the value of cc_claimrpt.AvailableReserves.

Dropping Reporting Tables

This version trigger drops tables for the following entities:

- FinancialsRptUpgrade
- TmpExposureRptStaging
- TmpStagingExposureRpt

After the database upgrade, ClaimCenter creates new tables for these entities upon server startup.

Converting Primary Key Indexes to Clustered Indexes

For SQL Server databases, this version trigger recreates non-clustered backing indexes for primary keys as clustered indexes. This change improves the performance of claim archiving and claim purging operations.

Before recreating the indexes, the version trigger automatically drops (and later rebuilds) any referencing foreign keys and drops any clustered indexes on tables with a primary key.

If you are using filegroups, the upgrade recreates the clustered index in the OP filegroup. By default, the upgrade also stores the intermediate sort results that are used to build the index in the OP filegroup. You can configure the upgrade to instead use the tempdb filegroup for the intermediate sort results.

If you want the upgrade to store the intermediate sort results in the tempdb filegroup, set the sqlserverCreateIndexSortInTempDB attribute of the upgrade element to true.

```
<database ...>
...
<upgrade sqlserverCreateIndexSortInTempDB="true" />
...
</upgrade>
</database>
```

This option increases the amount of temporary disk space that is used to create an index. However, it might reduce the time that is required to create or rebuild an index when tempdb is on a different set of disks from that of the user database.

By default, `sqlserverCreateIndexSortInTempDB` is `false`.

Renaming ClaimInfo Columns

This step renames `cc_ClaimInfo.ArchiveFailure` to `cc_ClaimInfo.ArchiveFailureID` and `cc_ClaimInfo.ArchiveFailureDetails` to `cc_ClaimInfo.ArchiveFailureDetailsID`.

Enforcing Contact Address Uniqueness

This trigger eliminates duplicate Address records in `ContactContact.Address` and `Contact.PrimaryAddress`. For duplicate Address records, the trigger creates a unique Address record and relinks the duplicate references to the unique Address.

Changing Datatype of ArchiveTransitionRec.InternalDesc

This step alters the datatype of the `cc_archivetransitionrec.internaldesc` column from `Text` to `MediumText`.

Changing Precision of Metric Percent Columns

This step alters the precision of the following metric percent columns to eight, with a scale of zero:

- `cc_claimmetric.PercentValue`
- `cc_claimmetriclimit.PercentRedValue`
- `cc_claimmetriclimit.PercentTargetValue`
- `cc_claimmetriclimit.PercentYellowValue`
- `cc_exposuremetric.PercentValue`
- `cc_exposuremetriclimit.PercentRedValue`
- `cc_exposuremetriclimit.PercentTargetValue`
- `cc_exposuremetriclimit.PercentYellowValue`

This trigger only runs if the `cc_claimmetric` table exists already, so it only runs if the database is being upgraded from ClaimCenter 6.0. For database upgrades from earlier versions, the metric percent columns are defined in the data model with the correct precision.

Adding Encryption to Claim Snapshots

ClaimCenter 6.0.8 provides encryption of sensitive data, such as tax IDs, on claim snapshots.

This step adds an `EncryptionVersion` integer column to `cc_claimsnapshot`. As of ClaimCenter 6.0.1, you can create multiple versions of `IEncryption` plugins. The `cc_claimsnapshot.EncryptionVersion` 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 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.

During the upgrade to ClaimCenter 6.0.8, an upgrade trigger increments the global encryption version number. So, when the Encryption Upgrade work queue runs, it will encrypt fields in the snapshot. This trigger does not fire if encryption is not enabled, and there are no encrypted fields defined on the claim snapshot.

For more information about the encryption plugin, see “Encryption Integration” on page 401 in the *Integration Guide*.

For more information about the Encryption Upgrade work queue, see “Batch Processes and Distributed Work Queues” on page 134 in the *System Administration Guide*.

Refactoring Claim and Exposure Staging Tables

This step drops the `ccst_claim` and `ccst_exposure` tables. ClaimCenter 6.0.2 and higher define `Claim.ClaimantDenorm`, `Claim.InsuredDenorm` and `Exposure.ClaimantDenorm` with the attribute `overwrittenInStagingTable` set to true. ClaimCenter automatically populates these columns when you load records into the staging tables. Previous versions of ClaimCenter populated these columns after the staging table records were moved into the main repository. In some cases, this caused performance issues. To remedy this, ClaimCenter 6.0.2 and higher populate these columns when you load the staging table rather than when you move data from the staging tables to the main repository.

Claim and exposure entities are defined as loadable, so ClaimCenter creates new `ccst_claim` and `ccst_exposure` tables when the server starts.

Cleaning up Cross-Claim ReserveLineWrapper.ReserveLine References

The trigger finds instances in which the `BulkInvoiceItemInfo` of a `ReserveLineWrapper` points to a different `Claim` than the `Claim` pointed to by the `ReserveLine`. The trigger updates those `ReserveLineWrapper` entities by repointing a given `ReserveLineWrapper` to a particular `ReserveLine` on its own `Claim`.

The trigger selects the `ReserveLine` belonging to the `Transaction` of the transferred Check. Typically, a `BulkInvoice` Check has only one `Transaction`. However, in this case, cross-claim linkage occurs when the `BulkInvoice` Check is transferred between `Claims`. In that case, the Check has two `Transactions`, an offset on the first `Claim` and an onset on the second. The trigger selects the onset `Transaction` on the original `Claim`.

Correcting Issues with Multiple Exposures on Incidents

If you attempted a database upgrade, and the upgrade reported issues with multiple exposures pointing to the same incident, follow instructions in this topic to resolve the issue. If you did not encounter this error, proceed to the next topic.

It is possible to have more than one exposure point to the same incident. This situation can potentially create a problem during upgrade. The database upgrade uses the `ExposureType` to determine which subtype to make the incident. If these multiple exposures have different `ExposureTypes` (as indicated by the `CoverageSubtype`) that are associated with different `IncidentTypes`, then the upgrade cannot determine the incident subtype. A version check reports this error. See “Remapping Incidents” on page 152.

If you modify the mapping of coverage subtypes to exposure types for ClaimCenter 6.0 to preserve your previous line of business configuration, you will not encounter this issue. The database upgrade will not need to recalculate incident subtypes. You can modify the mapping in Studio.

Otherwise, there are two approaches to handling this issue. The first approach recodes typecodes for each `ExposureType` and `CoverageSubtype` that is associated with an incident with multiple exposures. This affects every reference to the altered `ExposureTypes` and `CoverageSubtypes`, since the change is made in the typelist. The second approach adds a new typecode for each `ExposureType` and `CoverageSubtype` that is associated with an incident with multiple exposures. This typecode is used only for problem exposures. Review both approaches before proceeding.

The first approach only works if the `ExposureTypes` in question are not those with an ID of 11 or below. `ExposureTypes` with an ID of 11 or below are special and cannot be changed. If one of the following `ExposureTypes` needs to be changed, follow Approach 2.

- `BodilyInjuryDamage`
- `LossOfUseDamage`

- PersonalPropertyDamage
- PropertyDamage
- VehicleDamage
- LostWages
- WCInjuryDamage
- EmployerLiability
- GeneralDamage
- PIPDamages
- MedPay

Approach 1

Recode the existing ExposureType and CoverageSubtype typecodes so they do not collide with the new typecodes during upgrade. The general steps are:

1. Recode the typecodes in the pre-upgrade database.
2. Add the recoded typecodes to the target configuration as retired typecodes.
3. Upgrade the database.

After upgrading, existing exposures will have the recoded typecodes but new exposures will have the new typecodes. This might require you to update filters, searches, and any reporting that you are doing against exposures.

Note that this affects every reference to the altered ExposureTypes and CoverageSubtypes, since the change is made in the typelist.

To recode legacy typecodes

1. You tried to upgrade and the upgrade reported the multi-exposure error. This error provides a SQL statement that returns the ids of all Incident rows with multiple Exposures.
2. Determine which ExposureTypes and CoverageSubtypes are causing problems by running the following SQL queries:

```
SELECT DISTINCT exposuretype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
and
SELECT DISTINCT coveragesubtype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

3. Change the typecodes in the database. You can do this with two SQL statements, provided by database platform. This example adds the prefix old_ to the typecode:

Oracle:

```
UPDATE cctl_exposuretype
SET typecode = CONCAT('old_' + typecode)
WHERE id IN (<first sql from step 1>)
and
UPDATE cctl_coveragesubtype
SET typecode = CONCAT('old_' + typecode)
WHERE id IN (<second sql from step 1>)
```

4. Find the affected typecodes from the CoverageSubtype and ExposureType typelist configurations in the old configuration. Copy these into the target configuration.
5. In the target configuration, update the typecode field to match changes you made. Set these typecodes to retired so they do not appear in the user interface when making new exposures, and so forth.
6. Start the ClaimCenter 6.0.8 server to upgrade the database. The exposures that were a problem before do not change during the upgrade. The configuration in the new version for those CoverageSubtypes and ExposureTypes now looks the same as it did in the old version, besides the code field.

Approach 2

Add new typecodes in the pre-upgrade configuration for the problem exposures and then change only those exposures to use the new typecodes. Using this approach, only the problem exposures have an unorthodox ExposureType and CoverageSubtype. All other references to those typecodes will remain.

To recode legacy typecodes only for problem exposures

1. You tried to upgrade and the upgrade reported the multi-exposure error. This error provides a SQL statement that returns the ids of all Incident rows with multiple Exposures.

2. Determine which ExposureTypes and CoverageSubtypes are causing problems by running the following SQL queries:

```
SELECT DISTINCT exposuretype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

and

```
SELECT DISTINCT coveragesubtype FROM cc_exposure WHERE incidentid IN (<sql from the error>)
```

3. Find the problem ExposureTypes and CoverageSubtypes in the configuration files in the pre-upgrade configuration. Copy each one, giving the copy a new code, such as the existing code prefixed with old_.

4. Start the pre-upgrade server so that the new typecodes are added to the database.

5. Shut down the server.

6. Update the problem Exposure rows so that the CoverageSubtype and ExposureType refer to the new copies. For example:

```
UPDATE cc_exposure
SET exposuretype=<new exposure type id>
WHERE exposuretype=<old exposure type id>
AND incidentid IN (<sql from error>)
```

and

```
UPDATE cc_exposure
SET coveragesubtype=<new coverage subtype id>
WHERE coveragesubtype=<old coverage subtype id>
AND incidentid IN (<sql from error>)
```

Do this for each exposure type and coverage subtype that is causing the problem. Look up the id for the new types in the typelist tables (cctl_exposuretype and cctl_coveragesubtype).

7. Add the new typecodes to the ClaimCenter 6.0.8 target configuration files. You might want to retire them so that ClaimCenter does not display them.

8. Start the ClaimCenter 6.0.8 server to upgrade the database.

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.

Dropping Unused Columns on Oracle

By default, the ClaimCenter database upgrade on Oracle marks some columns as unused rather than dropping the columns. You can configure this behavior by setting the `oracleMarkColumnsUnused` parameter to `false` before running the database upgrade. This parameter is within the `<upgrade>` block of the `<database>` block of `config.xml`. If you set `oracleMarkColumnsUnused` to `false` before the upgrade, the upgrade already dropped removed columns. In that case, you do not need to perform the procedure in this section to drop unused columns.

Marking a column unused is a faster operation than dropping a column. Because these columns are not physically dropped from the database, the space used by these columns is not released immediately to the table and index segments. You can drop the unused columns after the upgrade during off-peak hours to free the space. ClaimCenter does not have to be shutdown to perform this maintenance task. You can drop all unused columns in one procedure, or you can drop unused columns for individual tables.

To drop all unused columns

1. Create the following Oracle procedure to purge all unused columns:

```
DECLARE
  dropstr VARCHAR2(100);
  CURSOR unusedcol IS
    SELECT table_name
    FROM user_unused_col_tabs;
BEGIN
  FOR tabs IN unusedcol LOOP
    dropstr := 'alter table '
              || tabs.table_name
              || ' drop unused columns';
    EXECUTE IMMEDIATE dropstr;
  END LOOP;
END;
```

2. Run the procedure during a period of relatively low activity.

To drop unused columns for a single table

1. Start the server to run the schema verifier. The schema verifier runs each time the server starts. If there are unused columns, the schema verifier reports a difference between the physical database and the data model. The schema verifier reports the name of each table and provides an SQL command to remove unused columns from each table.
2. Run the SQL command provided by the schema verifier. This command has the following format:

```
ALTER TABLE tableName DROP UNUSED COLUMNS
```

Setting Claim Descriptions

ClaimCenter 6.0 provides the option of using claim-level Insurance Services Office (ISO) messaging. Previous versions of ClaimCenter sent exposures to ISO. With ClaimCenter 6.0, you have the option of sending ISO messages at either the claim or exposure level. See “ISO Changes in ClaimCenter 6.0” on page 87 in the *New*

and Changed Guide.

New ISO validation rules require that a description is set on a claim if the claim is to be sent to ISO at the claim level. You will not be able to edit claims that lack a description until you set the description. Any batch processes that modify these claims or their objects, such as activities, will fail until you set the description. The failed batch process will report the following validation error:

```
[java] com.guidewire.pl.system.bundle.validation.EntityValidationException:
error:entity.Claim.Description/The claim's description must not be null
```

If a claim has already been sent to ISO at the exposure level, then it will never be sent at the claim level. Therefore, the description requirement only applies to claims that have not had any exposures sent to ISO.

If the following conditions are true, check for claims with null descriptions and add a description:

- You have had ISO enabled.
- You want to switch to claim-level ISO messaging.
- You have claims which have null descriptions and have not yet had any of their exposures sent to ISO.

You can check for claims that will cause this validation issue with the following SQL query:

```
SELECT claimnumber FROM cc_claim c
WHERE c.description IS NULL
AND NOT EXISTS (SELECT * FROM cc_exposure e
                WHERE e.ClaimID = c.ID AND e.ISOSendDate IS NOT NULL)
```

If this query returns any claim numbers, set a description for each corresponding claim.

Exporting Administration Data for Testing

Guidewire recommends that you create a small set of administration data from an upgraded data set. Use this data for development and testing of rules and libraries with ClaimCenter 6.0.8. This procedure is optional.

You might have already created an upgraded administration data set by following the procedure “Upgrading Administration Data for Testing” on page 234. If you followed that procedure, or you do not want an administration-only data set for testing purposes, you can skip this topic.

To create an administration data set for testing

1. Export administration data from your upgraded production database.
 - a. Start the ClaimCenter 6.0.8 server by navigating to ClaimCenter/bin and running the following command:


```
gwcc dev-start
```
 - b. Open a browser to ClaimCenter 6.0.8.
 - c. Log on as a user with the viewadmin and soapadmin permissions.
 - d. Click the **Administration** tab.
 - e. Choose **Import/Export Data**.
 - f. Select the **Export** tab.
 - g. For **Data to Export**, select **Admin**.
 - h. Click **Export**. Your browser will note that you are opening a file and will prompt you to save or download the file.
 - i. Select to download the admin.xml file.
2. Create a new database account for the development environment on a database management system supported by ClaimCenter 6.0.8. 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 <http://guidewire.custhelp.com>.

See “Configuring the Database” on page 20 in the *Installation Guide* for instructions to configure the database account.

3. Install a new ClaimCenter 6.0.8 development environment. Connect this development environment to the new database account that you created in step 2. See the *ClaimCenter Installation Guide* for instructions.
4. Copy the `admin.xml` file that you exported to a location accessible from the new development environment.
5. Open `admin.xml`.
6. Modify each occurrence of `cc_systemTables` to `systemTables`.
7. Modify the `public-id` of the default organization from `<Organization public-id="default_data:1">` to `<Organization public-id="systemTables:1">`
8. Update all references to the default organization to the new `public-id` by changing each occurrence of `<Organization public-id="default_data:1"/>` to `<Organization public-id="systemTables:1"/>`.
9. Change the root group `public-id` from `<Group public-id="default_data:1">` to `<Group public-id="systemTables:1">`.
10. Change all references to the root group on group users and assignable queues by changing each occurrence of `<Group public-id="default_data:1"/>` to `<Group public-id="systemTables:1"/>`.
11. Change all references to the root group on child groups by changing each occurrence of `<Parent public-id="default_data:1"/>` to `<Parent public-id="systemTables:1"/>`.
12. Save changes to `admin.xml`.
13. Create an empty version of `importfiles.txt` in the `modules/configuration/config/import/gen` directory of the new development environment.
14. Create empty versions of the following CSV files:
 - `activity-patterns.csv`
 - `authority-limits.csv`
 - `reportgroups.csv`
 - `roleprivileges.csv`
 - `rolereportprivileges.csv`Leave `roles.csv` as the original complete file.
15. Import the administration data into the new database:
 - a. Start the ClaimCenter 6.0.8 development server by navigating to `ClaimCenter/bin` and running the following command:
`gwcc dev-start`
 - b. Open a browser to ClaimCenter 6.0.8.
 - c. Log on as a user with the `viewadmin` and `soapadmin` permissions.
 - d. Click the **Administration** tab.
 - e. Choose **Import/Export Data**.
 - f. Select the **Import** tab.
 - g. Click **Browse**.
 - h. Select the `admin.xml` file that you exported from the upgraded production database and modified.
 - i. Click **Open**.

Final Steps After The Database Upgrade is Complete

This section describes the checks and procedures to run after you have completed the upgrade procedure and migration of configurations and integrations. The processes and checks in this section provide you with a benchmark of the new system. Completing these steps is particularly important to going live in a production environment.

IMPORTANT For procedures referenced below that use utilities in `toolkit/bin` on the pre-upgrade configuration, use the utility from `admin/bin` in the target configuration following the upgrade. You do not need to generate the toolkit in the target configuration to use these utilities.

Use these procedures to revalidate the database:

- “Validating the Database Schema” on page 238
- “Checking Database Consistency” on page 238
- “Creating a Data Distribution Report” on page 239
- “Generating Database Statistics” on page 240
- “Running Key Batch Processes” on page 290
- “Backing up the Database After Upgrade” on page 291

Running Key Batch Processes

Before going live, run the following batch processes while the server is at the MAINTENANCE run level:

- `dashboardstatistics`
- `financialscalculations`
- `claimhealthcalc` (optional)

To set the server run level to maintenance, use the following command:

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

To run a batch process, use the `admin/bin/maintenance_tools` command:

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

These batch processes update the user dashboard, team statistics, financial data and claim metrics in the system. For a detailed explanation of each batch process, see “Batch Processes and Distributed Work Queues” on page 134 in the *System Administration Guide*.

For more information on the `maintenance_tools` command, see “`maintenance_tools` Command” on page 171 in the *System Administration Guide*.

Calculating Claim Health Metrics

Claim health metrics is a new feature in ClaimCenter 6.0. For more information on this feature, see “Claims Performance Monitoring” on page 26 in the *New and Changed Guide* and “Claim Performance Monitoring” on page 315 in the *Application Guide*.

If you want to use claim metrics, and you want the threshold to apply to existing claims, configure metrics and run the `claimhealthcalc` batch process. If you do not want to use claim metrics, or you do not want metrics to apply for existing claims, you do not need to run the `claimhealthcalc` batch process.

Before you run `claimhealthcalc`, configure claim and exposure metrics for your business. See “Administering Metrics and Thresholds” on page 322 in the *Application Guide*.

Upgraded claims and exposures do not have metrics calculated. Run the `claimhealthcalc` batch process to calculate metrics. Some claims and exposures will have metrics that are already yellow or red. For example, one metric tracks how many days a claim has been open. By default, this metric turns yellow after four days and red

after six days. The `claimhealthcalc` batch process calculates this and other metrics for each claim. Metric data includes information on when each metric reached yellow and red status. However, after you upgrade and run `claimhealthcalc`, metrics in yellow or red status have a reach time of when the batch process was run. For example, a claim was opened on October 1, is not closed, and you run `claimhealthcalc` on October 10. The claim meets the yellow and red thresholds for the days open metric. The `claimhealthcalc` batch process sets `ClaimMetric.ReachYellowTime` and `ClaimMetric.ReachRedTime` to the time the batch process runs, rather than October 5 and October 7 respectively.

Backing up the Database After Upgrade

Finally, before going live, back up the upgraded database. This provides you with a snapshot of the initial upgraded data set, if an unanticipated event occurs just after going live.

Upgrading Integrations and Gosu from 4.0.x

This topic lists the tasks to upgrade Gosu code and integration points to this release. The tasks are presented in tables, according to when you perform the tasks. You can print these tables to use them as checklists during the upgrade.

This topic includes:

- “Overview of Upgrading Integration Plugins and Code from 4.0.x” on page 293
- “Tasks Required Before Starting the Server” on page 294
- “Tasks Required Before Deploying a Production Server” on page 297
- “Tasks Required Before the Next Upgrade” on page 299

Overview of Upgrading Integration Plugins and Code from 4.0.x

This topic provides a high level approach to upgrading integration plugins and code. Review this topic, then proceed to the following topics for specific upgrade steps:

- Tasks Required Before Starting the Server
- Tasks Required Before Deploying a Production Server
- Tasks Required Before the Next Upgrade

As part of integration, developers add third-party libraries (JARs) to the toolkit libraries to compile their code. During the upgrade phase, segregate third-party libraries from the toolkit libraries. Initially, it is more practical to make use of these third-party libraries as is during the upgrade process. Later, you can upgrade these libraries separately, along with any ramification to the code, as necessary.

The database upgrade usually matures over the initial cycles of the upgrade process. If the integration code upgrade starts at the same time, regenerating the toolkit might not yield the final version of the toolkit libraries. Consult with the database upgrade team to determine when to regenerate the toolkit for more current libraries.

Integration upgrade steps

1. Create a project with the code at hand after segregating the toolkit libraries from third-party libraries.
2. Ensure you can successfully compile.
3. Create a backup copy of the project.
4. Replace the toolkit libraries with the upgraded toolkit libraries. Leave third-party libraries as is.
5. Update to the correct version of Java. See “Installing Java” on page 27 in the *Installation Guide*.
6. Many classes will fail to compile correctly. The error list is literally the technical upgrade. It needs to be sorted and addressed.

The most commonly encountered compiler failures during upgrade are described in the following table:

Issue	Example	How to approach upgrade
Java upgrades	Refer to Java documentation.	
Changes in object construction	<code>EntityFactory.getEntityFactory().newEntity()</code> → <code>EntityFactory.getInstance().newEntity()</code>	Identify the changes then use a utility to find and replace throughout the code base.
Name changes	<code>gscrip</code> → <code>gosu</code>	Identify the changes then use a utility to find and replace throughout the code base.
Discontinued support of utilities available in previous versions	<code>com.guidewire.util.FileSystem</code> and <code>com.guidewire.util.FileUtil</code>	Carry the old implementation forward as a third-party library.
Class relocation	<code>com.guidewire.logging.SystemOutLogger</code> → <code>gw.util.SystemOutLogger</code>	Locate the new package using searches or a utility such as <code>scanzip</code> in the new toolkit.
Additional interface methods	The <code>IDocumentContentSource</code> interface gained additional methods: <ul style="list-style-type: none"> • <code>getDocumentContentsInfoForExternalUse()</code> • <code>isInboundAvailable()</code> • <code>isOutboundAvailable()</code> 	Review the <i>ClaimCenter New and Changed Guide</i> for additional methods on key interfaces used in your integration plugins.
Functional changes (most involved to upgrade)	Relation between <code>Claim</code> and <code>BodyPartDetails</code> and <code>BodyPartType</code> changing to include <code>InjuryIncident</code> Using a <code>RiskUnit</code> such as <code>VehicleRU</code> or <code>PropertyRU</code> instead of <code>PolicyVehicle</code> or <code>PolicyProperty</code>	Understand the changes and what the code is trying to do and modify your code accordingly. Review database upgrade triggers to understand data model changes. Review the <i>ClaimCenter New and Changed Guide</i>

Tasks Required Before Starting the Server


The following table contains things you must do before you start the server.



Tasks


For more information...

 Tasks	For more information...
Tasks related to upgrading from ClaimCenter 4.0 to ClaimCenter 5.0	
<input type="checkbox"/> Follow the configuration upgrade procedure.	"Upgrading the ClaimCenter 4.0.x Configuration" on page 175
<input type="checkbox"/> Follow the database upgrade procedure.	"Upgrading the ClaimCenter 4.0.x Database" on page 233
<input type="checkbox"/> Confirm your Java classes were moved to the correct locations that changed due to the new modules feature. You must also update your Java development environment to generate Java class files or libraries to the new locations.	"Plugins Directories Moved In Relation to Config Directory" on page 153 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review the new Plugins editor in Studio and confirm your plugin settings upgraded properly. Additionally, if you use ContactCenter, there is a special required procedure.	"Plugin Registration Changes" on page 155 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you used built-in demonstration plugins or default plugin implementations, review your code for package names that may have changed.	"Plugin Built-in Implementation Classes Package Changes" on page 155 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Change older GScript plugins to implement a GScript interface.	"GScript Plugin Important and Required Changes" on page 156 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Change older GScript plugins to use new-style plugin parameter syntax.	"GScript Plugin Important and Required Changes" on page 156 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you used SOAP plugins (plugin implementations that call a remote SOAP service), convert them to GScript plugins that call out to your SOAP API for each method.	"SOAP Plugins No Longer Needed" on page 158 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use the deprecated assignment plugin, rewrite your code to instead use the new assignment APIs.	"IAssignmentAdapter Deprecated" on page 159 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use the deprecated export tools web service interface, do one of the following instead. Either use the user interface to export your administrative data, or use a custom web service API that sends necessary data one record at a time.	"IExportTools Removed" on page 159 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you used the included MapPoint plugin unmodified, reconfigure your server to use the new included MapPoint plugin that is implemented in GScript and has a different class symbol.	"Geocoding Plugin Changes" on page 160 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you customized geocoding plugin Java files, either merge your changes as GScript code into the new geocoding plugin or use an included temporary (until next upgrade) plugin wrapper.	"Geocoding Plugin Changes" on page 160 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Optional. If you have many document templates, consider enabling the new document template descriptor caching feature for better application server and user interface performance.	"Document Management Changes" on page 163 in the <i>New and Changed Guide</i>
<input type="checkbox"/> To disable additional document templates or descriptors added to your configuration module, delete associated files in that module.	"Document Management Changes" on page 163 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Verify the document management directory settings used by content source and metadata source plugins.	"Document Management Changes" on page 163 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure your reporting tools do not rely on the removed message state called pending retry.	"Message State and Message History Changes" on page 164 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Confirm your messaging plugins do not rely on old transaction behavior between different phases of messaging plugin method calls, particularly in error and exception handling.	"Messaging Transaction Changes and MessageRequest Plugin" on page 165 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure you do not use old-style message sinks, converting to new messaging plugins if necessary.	"Message Sinks Now Unsupported" on page 166 in the <i>New and Changed Guide</i>

 Tasks	For more information...
<input type="checkbox"/> Confirm your transaction public ID field is has appropriate length, at least two or more characters shorter than the maximum length.	"Transaction Public ID Length Must Be Shorter" on page 166 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure you do not use multi-registration of plugins, which was rarely used and is now unsupported.	"Multi-Registration of Plugins for Wrapping Unsupported" on page 166 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure there are no syntax errors in Studio due to minor change in meaning of the Java list class in GScript code in unusual cases.	"Minor Change In Meaning of 'List'" on page 141 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Convert all references to a typekey name property to the new localized name field. The localized name field displays in the local language. In rare cases, convert the typekey name property reference to the unlocalized name field, for debugging or compatibility with older code.	"Typekey.Name Becomes DisplayName and UnlocalizedName" on page 142 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure all references to the deprecated TypeKeys property of a type are converted to the new method syntax.	"Type.TypeKeys Property Becomes Method with Argument" on page 142 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove usages of the toMap method on collections. Do not reintroduce a method with the same name because Guidewire reserves that method name on collections for future use.	"Removal of toMap Method on Collection" on page 143 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Set the config.xml file parameter SortWSDLsUponStudio-Import to true to enforces backward compatibility with how GScript handles remote web services registered in Studio. Set it to false for the new behavior, but that requires careful review of APIs that call remote web services for possible change in function argument order.	"MapPoint Updates for New WSDL Parameter Ordering" on page 168 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you want to to use the new WSDL behavior with correct (fixed) parameter ordering, make one small change to the built-in MapPoint implementation to avoid a compile error. This change is unnecessary if you set the vallue of Sort-WSDLsUponStudioImport to true, which enforces backward compatibility.	"MapPoint Updates for New WSDL Parameter Ordering" on page 168 in the <i>New and Changed Guide</i>


Tasks related to upgrading from ClaimCenter 5.0 to ClaimCenter 6.0


<input type="checkbox"/> If you modified built-in Gosu enhancements to lists, update your code to merge changes into the new package location	"Moved Enhancement Packages" on page 63 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Check for compile errors due to the new requirement that Gosu blocks cannot contain single statements. To fix this, add braces around the statement. This does not affect expressions as the body of a Gosu block.	"Blocks Require Single Statements in Braces" on page 66 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you wrote any Gosu enhancements on arrays, update the enhancement definition file with the updated syntax.	"Gosu Array Enhancement Changes" on page 72 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency, check for compilation errors in math operations that mix currency amounts and big decimal values. Refactor the code to use currency amounts to preserve the currency information.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove any references to previously-deprecated and now-removed message methods getKey and addKey. Replace with the renamed APIs.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review messaging code and any 'find' queries that reference the message property MessageSink. Update to instead use new name DestinationID.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Check messaging code for compilation errors due to removed message context method for set and get destination and replace with renamed methods.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>

 Tasks	For more information...
<input type="checkbox"/> Check for compilation errors related to package changes of document management plugins and plugin implementations.	"Document Management Plugin Implementation Location Changes" on page 101 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review Java code for compilation errors that require minor change related to stripping parameterization of types in external entities.	"Parameterization of Types Stripped from Java External Entities" on page 101 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you used the validation plugin, refactor to use the Validation rule set instead.	"Validation Plugin Removed" on page 96 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update your development build system to copy the entity libraries JAR file to your configuration module after regenerating it.	"Copy the Plugin Entity Library to Your Configuration Module" on page 86 in the <i>New and Changed Guide</i>
<input type="checkbox"/> For document management storage plugins, add new properties (for Gosu) or methods (for Java) for inbound and outbound method availability.	"Document Management Plugin Changes for Availability" on page 103 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Updating geocoding plugin with a new method signature to get maps with options.	"Geocoding Plugin Changes for Map Options" on page 104 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Fix any compile errors related to changes in function pointers.	"Function Pointers and Nested Functions Now Unsupported" on page 74 in the <i>New and Changed Guide</i>

Tasks Required Before Deploying a Production Server

The following table contains tasks to complete before starting the server and changes to familiarize yourself with before deploying a server to a production environment.


 Tasks	For more information...
Tasks related to upgrading from ClaimCenter 4.0 to ClaimCenter 5.0	
<input type="checkbox"/> Review code for case issues, particularly for performance-intensive code. This results in faster GScript performance and lower memory usage.	"GScript Case Sensitivity Implications" on page 140 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Optional. Review web service interfaces for security settings to determine whether to suppress any web services completely or change required permissions for a remote user to use to the service.	"Web Service Visibility and Customizability Changes" on page 154 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you used the included MapPoint plugin unmodified, reconfigure your server to use the new included MapPoint plugin that is implemented in GScript and has a different class symbol.	"Geocoding Plugin Changes" on page 160 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure your integration code does not rely on the assignment entity, which is now gone. Instead use the new assignment data model and APIs to detect changes to assignments directly on assignable entities.	"Ensure Messaging Code Does Not Rely on Assignment Entity" on page 165 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review uses of the type list method getByCode to ensure you do not rely on ignoring retired typecodes.	"TypeList.getByCode() Changes" on page 143 in the <i>New and Changed Guide</i>
Tasks related to upgrading from ClaimCenter 5.0 to ClaimCenter 6.0	
<input type="checkbox"/> Review your code for any String literals and Gosu templates that contain \${...} syntax. If you find any, refactor to avoid accidental use of new Gosu String templates, and test your new code.	"Gosu Templates" on page 54 in the <i>New and Changed Guide</i>

 Tasks	For more information...
<input type="checkbox"/> Review your code for ternary condition expressions (condition ? trueValue : falseValue). Check for any cases in which you pass different types to the true clause and the false clause. If you did not declare the variable type, check for possible changes to the compile-time type of the result.	"Compound Types" on page 62 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review all uses of the run with new bundle API. In the rare case that you use this API in code with a current user, it throws an exception before your block runs. Ensure your code does not rely on the old behavior in this unusual case.	"Exception Changes If No Current User and Creating New Bundle" on page 74 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update any tools or scripts in your build system to use the new locations of scripts and Java libraries.	"Changes to Structure of Integration-related Files and Scripts" on page 85 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Decide whether you want to use new optional claim-based ISO sending. Also, review all ISO code for changes in this release and update your code to use the new payload generation classes. It is possible to use your old payload generation code in this release, but it is best to convert your code to the new system.	"ISO Changes in ClaimCenter 6.0" on page 87 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency and the initial reserves plugin, refactor to use Initial Reserves rule set instead.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency and the policy search plugin, update your plugin to set the currency for a policy.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use multicurrency, review your code that manipulates currency amounts and big decimal values. Refactor the code to use currency amounts to preserve the currency information.	"Multicurrency-related Changes" on page 97 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review code relating to handling duplicate messages and message history. Ensure your code relies on the new behavior.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review messaging code to update method signature for the find by sender reference ID method.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Convert uses of the client-side FNOL mapper to the new server-side FNOL mapper system.	"FNOL Mapper is Now a Server-Side Tool" on page 94 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update any integration code that uses the claim API web service to handle the archiving method name change, and other changes listed in the What's New and Changed book.	"IClaimAPI Web Service Interface Changes" on page 103 in the <i>New and Changed Guide</i>

Tasks Required Before the Next Upgrade

The following table contains tasks required before the next upgrade. For example, if you used APIs that are now deprecated, begin rewriting your code to avoid use of deprecated APIs. Guidewire will remove these APIs in a future release.

 Tasks	For more information...
<input type="checkbox"/>	
<input type="checkbox"/> Tasks related to upgrading from ClaimCenter 4.0 to ClaimCenter 5.0	
<input type="checkbox"/> Convert GScript class constructors to the new constructor style.	"GScript Constructor Syntax Changed" on page 137 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure you address all GScript syntax errors Studio indicates due to stricter requirements for all code paths.	"Stricter Return Statements Requirements for All Code Paths" on page 138 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Optional. Review web service interfaces to determine whether to suppress generation in the toolkit for any web services for performance reasons of toolkit regeneration during development. This setting never affects run time performance nor run time security.	"Web Service Visibility and Customizability Changes" on page 154 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Optional. Review Java plugin implementations to determine if some might be better as GScript plugins to take advantage of new GScript and Studio features, such as native web service support.	"GScript Plugin Important and Required Changes" on page 156 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure you remove uses of the deprecated entity factory class in GScript. Java usages of the class do not need changes.	"EntityFactory Deprecated in GScript (Not in Java)" on page 139 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Ensure you remove references to the itype property on objects and other types. Use the typeof operator instead.	"Deprecation of itype Property" on page 141 in the <i>New and Changed Guide</i>
Tasks related to upgrading from ClaimCenter 5.0 to ClaimCenter 6.0	
<input type="checkbox"/> Review your code related to resource management, streams, and locking. Look for places in which the new using clauses might make code easier to understand or safer.	"Object Lifecycle Management with the 'using' Keyword" on page 56 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code to remove unnecessary downcasting. This makes your Gosu code easier to read and maintain.	"Type Inference Downcasting" on page 56 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review your code to use the new compact object initializer syntax. This makes your Gosu code easier to read and maintain.	"Object Initializer Syntax During Object Creation" on page 58 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Change any references to gw.api.xml.XMLNode to instead use gw.xml.XMLNode.	"XML Node Package Name Changed" on page 63 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you use the old-style XSD registration system (including registry-xsd.xml), update your code to use the new XSD class loading system. This avoids your Gosu code using now-deprecated types.	"Changes to XSD Class Loading Behavior" on page 64 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Change all uses of deprecated collection methods to the renamed versions. Affects: countMatches, findFirst, findAll, findByType, removeMatches, keepMatches. The replacement for findFirst changes behavior with empty sets. While renaming methods, carefully review all code that uses findFirst to ensure you do not rely on the old behavior.	"Changes to Existing Collections Enhancement Methods" on page 67 in the <i>New and Changed Guide</i>

 Tasks	For more information...
<input type="checkbox"/> Review your code for uses of the now-deprecated list/array expansion operator (the single period). Replace these with the new *. operator syntax.	"New Array/List Expansion Operator (Deprecated Old Style)" on page 71 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove any deprecated variable scoping modifiers and replace with new scoping and concurrency APIs.	"New Concurrency and Scoping APIs, Scoped Variables Deprecated" on page 73 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update any Gosu deprecated references to the plugin registry class. This does not affect Java code.	"New Plugin Registry (Old Version Deprecated in Gosu)" on page 101 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update your code to use custom data model extensions to replace deprecated message properties for message code and optional String data.	"Messaging API Changes" on page 98 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Review PCF pages, libraries and rules for deprecated usages of Contact.Person and similar casting accessors on the Contact object. Update as necessary.	"Contact Casting Accessors Deprecated" on page 40 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Convert all Gosu template code to use the new template system. If you need to run templates from web services, write custom web services unique to each integration point.	"Gosu Templates" on page 54 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If you need to run templates from web services, write custom web services unique to each integration point.	"Gosu Templates" on page 54 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Find deprecated references to type properties and change your code to access those properties directly on the type, not the original object.	"New 'Type' Property on All Types" on page 73 in the <i>New and Changed Guide</i>
<input type="checkbox"/> If any integration code used the template execution web service (the data extraction API), convert these cases to custom web services for each integration point.	"IDataExtractionAPI Web Service Interface Deprecated" on page 102 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update external integration code that uses the messaging tools API method for acknowledging a message, and use the new version.	"IMessagingToolsAPI Web Service Interface Changes" on page 102 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Remove any uses of the deprecated segmentation plugin and replace with rule set code.	"Segmentation Plugin Deprecated" on page 107 in the <i>New and Changed Guide</i>
<input type="checkbox"/> Update financials integration code to use the new domain methods on financials objects, rather than the deprecated message-based methods.	"Financials Integration APIs for Acknowledgement-based Transitions" on page 95 in the <i>New and Changed Guide</i>