Guidewire ClaimCenter®

Configuration Upgrade Guide

RELEASE 1.11.0

GUIDEWIRE
Adapt and succeed™

Product Name: Guidewire ClaimCenter Configuration Upgrade Tool
Product Release: 1.11.0
Document Name: ClaimCenter Configuration Upgrade Guide
Document Revision: 14-July-2016

Contents

About ClaimCenter Documentation

The following table lists the documents in ClaimCenter documentation.

Document	Purpose
InsuranceSuite Guide	If you are new to Guidewire InsuranceSuite applications, read the *InsuranceSuite Guide* for information on the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with Guidewire applications.
Application Guide	If you are new to ClaimCenter or want to understand a feature, read the *Application Guide*. This guide describes features from a business perspective and provides links to other books as needed. The intended readers are everyone who works with ClaimCenter.
Upgrade Guide	Describes the overall ClaimCenter upgrade process, and describes how to upgrade your ClaimCenter database from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing ClaimCenter application extensions and integrations.
Configuration Upgrade Guide	Describes how to upgrade your ClaimCenter configuration from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing ClaimCenter application extensions and integrations. The *Configuration Upgrade Guide* is published with the Upgrade Tools, and is available on the Guidewire Resource Portal.
New and Changed Guide	Describes new features and changes from prior ClaimCenter versions. Intended readers are business users and system administrators who want an overview of new features and changes to features. Consult the “Release Notes Archive” part of this document for changes in prior maintenance releases.
Installation Guide	Describes how to install ClaimCenter. The intended readers are everyone who installs the application for development or for production.
System Administration Guide	Describes how to manage a ClaimCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
Configuration Guide	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files. The intended readers are all IT staff and configuration engineers.
PCF Reference Guide	Describes ClaimCenter PCF widgets and attributes. The intended readers are configuration engineers.
Data Dictionary	Describes the ClaimCenter data model, including configuration extensions. The dictionary can be generated at any time to reflect the current ClaimCenter configuration. The intended readers are configuration engineers.
Security Dictionary	Describes all security permissions, roles, and the relationships among them. The dictionary can be generated at any time to reflect the current ClaimCenter configuration. The intended readers are configuration engineers.
Globalization Guide	Describes how to configure ClaimCenter for a global environment. Covers globalization topics such as global regions, languages, date and number formats, names, currencies, addresses, and phone numbers. The intended readers are configuration engineers who localize ClaimCenter.
Rules Guide	Describes business rule methodology and the rule sets in ClaimCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.

Document	Purpose
Contact Management Guide	Describes how to configure Guidewire InsuranceSuite applications to integrate with ContactManager and how to manage client and vendor contacts in a single system of record. The intended readers are ClaimCenter implementation engineers and ContactManager administrators.
Best Practices Guide	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
Integration Guide	Describes the integration architecture, concepts, and procedures for integrating ClaimCenter with external systems and extending application behavior with custom programming code. The intended readers are system architects and the integration programmers who write web services code or plugin code in Gosu or Java.
Java API Reference	Javadoc-style reference of ClaimCenter Java plugin interfaces, entity fields, and other utility classes. The intended readers are system architects and integration programmers.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Gosu API Reference	Javadoc-style reference of ClaimCenter Gosu classes and properties. The reference can be generated at any time to reflect the current ClaimCenter configuration. The intended readers are configuration engineers, system architects, and integration programmers.
Glossary	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.

Conventions in This Document

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

Support

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

part I

Upgrade Overview

chapter 1

Planning Your ClaimCenter Upgrade

Before you begin your upgrade, download the latest version of the Configuration Upgrade Tools, along with any associated documentation updates.

Review the Configuration Upgrade Tools release notes and any updated documentation. The latest versions of these resources are available on the Guidewire Resource Portal – `http://guidewire.custhelp.com`.

IMPORTANT Guidewire recommends that you consult with Guidewire Services before beginning an upgrade. Guidewire Services has a number of Knowledge Base articles and Accelerators that might assist with your 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 a ClaimCenter upgrade to version 9.0. Read this topic before beginning an upgrade.

IMPORTANT Review the following topics before beginning the upgrade procedure:

- “New and Changed in 9.0.0” in the *New and Changed Guide*.
- “New and Changed in 8.0.0” in the *New and Changed Guide*.
- “Release Notes Archive” in the *New and Changed Guide* for changes to maintenance releases between major versions.
- “Upgrade Issues” in the ClaimCenter 9.0 release notes for issues specific to this release. If there are no upgrade issues specific to ClaimCenter 9.0, there is not an “Upgrade Issues” topic in the release notes.

- If you have any language packs installed, see “Upgrading Display Languages” in the *Globalization Guide*.

The upgrade procedure is presented in three topics: upgrading the configuration, upgrading the database, and upgrading Gosu code and integration points.

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

If upgrading ClaimCenter 8.0, see the topics in “Upgrading from 8.0” on page 29.

If upgrading ClaimCenter 7.0, see the topics in “Upgrading from 7.0” on page 53.

If upgrading an earlier release of ClaimCenter, review the information in “Upgrading from Versions Prior to 7.0” on page 14.

This topic includes:

Overview of the Upgrade Process

How Guidewire Architecture Supports Upgrades

The Guidewire approach to building enterprise software includes an upgrade path to future releases without ever modifying core product source code on a per-customer basis. To make this approach possible, Guidewire supports a broad envelope of configuration. Customers can override base content or add entirely new content through configuration. This is done in a way that externalizes the configuration from the core product.

All customers run generally available releases of Guidewire products and use the extensive capability to configure the application. Each application is built on top of the Guidewire platform, which exists to provide a common technology layer to common to all InsuranceSuite applications. This platform enables the application-specific logic and configuration to be completely separate from the platform so that changes to each can be made independently. For example, the platform layer provides the technology and tools for rendering the user interface, but the user interface design and layout reside in the application layer:

Customer configurations of base functional content are stored separately from the base content in a separate configuration layer; the base content is never modified by customer projects. This enables customers to make changes that are specific to their implementations and independent of the application layer. As a result, customer-specific configuration is not interwoven with the application layer. This simplifies the upgrade process and enables customer configuration changes to survive the upgrade process.

Using the Guidewire Studio environment, customers can modify all screens in the application, delete screens, and define new screens. The data model can be extended, with any extensions available for use in the UI, business rules, and integration code. With Studio, business rules can be created, modified, and removed. The full range of data in each application is exposed for integration purposes, enabling customers to integrate Guidewire applications with a wide range of external systems. Integration code can also be developed in the Guidewire Studio environment or externally through a Java IDE, such as Eclipse.

With customer configuration externalized from the core product, it can be carried forward at the point of a major upgrade. Guidewire provides tools that help identify and handle changes and conflicts during an upgrade and assist with the automation of the upgrade process.

The main effort during an upgrade is the upgrading of resources that were modified by customers as part of their implementation and also modified by Guidewire in the new release. The effort needed can be greatly mitigated by adhering to best practices during implementation. Specifically, it is helpful to ensure that potential conflicts are not only minimal but also easily identifiable as configuration changes belonging to the customer. This means following recommended naming conventions for customer extensions, as well as commenting and segregating customer code to minimize the risk of conflict with future Guidewire changes.

Upgrades Prior to Initial Production Deployment

If a new major version is released during an initial implementation project, it is common to perform a mid-implementation upgrade. By enabling the implementation to go live on the more current version, this type of upgrade has many advantages, such as:

- Mid-implementation upgrades take less time and have less risk than production upgrades. There is no need for regression testing against a copy of a production database, and there is also no need for a production database upgrade. The project team will leverage the current implementation's Stabilization Phase for regression testing of the upgrade work.
- If the initial development is still in progress, there will simply be less code to upgrade.
- The deployment will benefit from a longer standard support period with the latest Guidewire product and required third party database and application server versions.
- The project can leverage new business and technical features from the latest version.

Guidewire strongly recommends that customers consider including a major version upgrade as part of their initial implementation if the timing of the major release aligns with the implementation project. Your Guidewire project manager can provide an estimate for your mid-implementation upgrade based on how far along you are in your implementation.

Overview of ContactManager Upgrade

The automatic upgrade process for ContactManager is almost precisely the same as for ClaimCenter. However, there are differences, especially for manual upgrade. Additionally, Guidewire recommends that you complete the ContactManager upgrade before manually updating files that ClaimCenter uses for integration with ContactManager.

- For information on upgrading ContactManager, see "Upgrading ContactManager from 7.0" in the *Upgrade Guide*.
- For information on upgrading ClaimCenter files used to integrate with ContactManager, see "Upgrading ClaimCenter from 7.0 for ContactManager" in the *Upgrade Guide*.

Supported Starting Version

You can only upgrade an Oracle or SQL Server database. Guidewire does not support upgrade of the H2 Quick-start development database.

Upgrading from 7.0

You can upgrade to ClaimCenter 9.0 directly from any ClaimCenter 7.0 version.

Upgrading from 8.0

You can upgrade to ClaimCenter 9.0 directly from any ClaimCenter 8.0.1 or later version.

Upgrading from Versions Prior to 7.0

You cannot upgrade to ClaimCenter 9.0 directly from ClaimCenter versions prior to 7.0. You must first upgrade your data model and database to the most recent ClaimCenter 8.0 release, as follows:

To upgrade your database to 8.0, merge data model files (entities, typelists, field validators and data types), and correct errors that prevent the 8.0 server from starting. For the production deployment execution, you only need a 8.0 batch server and a 8.0-compatible database. See the *Guidewire Platform Support Matrix* for 8.0 database requirements. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at `https://guidewire.custhelp.com/app/resources/products/platform`.

Since the 8.0 application will never be used interactively by any users, you do not need to upgrade your entire configuration and integrations to 8.0, only the data model.

It is not necessary to upgrade your entire custom configuration to 8.0 and then test the entire application in 8.0 before upgrading to 9.0. This process would add several months to the project duration with no significant benefit.

To upgrade to 8.0

1. Upgrade all data model files (entities, typelists, field validators, data types).

   For **BOTH_ADD** files – Merge Guidewire changes with your custom changes, generally keeping both sets of changes.

   For **BOTH_EDIT** files – Merge Guidewire changes with your custom changes, generally keeping both sets of changes.

   For **EDIT_DELETE** files – Keep your custom version of the file, but make note of the files that are in this category. These files appear under the **CUSTOMER_ADD** filter during the 9.0 merge process. You will need to reevaluate the differences and decide whether to accept the deletion of the file when merging into 9.0.

   For all other categories of files – Accept all Guidewire changes and accept all your custom changes.

2. Upgrade all other (non data model) files:

   For **BOTH_ADD** files – Keep only your custom version of the file.

   For **BOTH_EDIT** files – Keep only your custom version of the file.

   For **EDIT_DELETE** files – Keep your custom version of the file, but make note of the files that are in this category. These files appear under the **CUSTOMER_ADD** files during the 9.0 merge process. In most cases, you will not want to accept the file into the 9.0 configuration. Instead, you will rebuild the desired logic in the appropriate files for the 9.0 release.

   For all other categories of files – Accept all Guidewire changes and accept all your custom changes.

The process described above will minimize the effort associated with upgrading the database to 8.0, because only the data model files must be merged. This is the minimum work necessary to be able to start the 8.0 server and trigger the automated database upgrade to 8.0. All remaining (non data model) files need only be merged once, into the 9.0 release.

Upgrading Language Packs

This document does not provide instructions for upgrading language packs. See "Upgrading Display Languages" in the *Globalization Guide* before continuing with the ClaimCenter upgrade. Also see the release notes included with the target release version of the language pack.

IMPORTANT If your base release has a language pack installed, you must uninstall the language pack prior to upgrading.

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 9.0. 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. Configuration upgrade 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.

The time required to upgrade your configuration depends on the complexity of the installation and resources available to perform the upgrade. If you choose to deploy new features in a release, implementing them can also impact the upgrade 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.
- Review “Upgrading Display Languages” in the *Globalization Guide* if upgrading with a language pack installed.
- 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 training about changes and new features in 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 upgrade to ClaimCenter 9.0 require 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:

- 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. Test upgrading a copy of production data as early as possible when the upgrade includes LOB typelist changes. This provides time to address issues that might arise with production data that would not otherwise be detected.
- 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.

Sample Deployment Plan

This topic includes a sample deployment plan for the upgrade of ContactManager and ClaimCenter.

ContactManager Prerequisite steps

1. Complete the entire configuration upgrade (code merge) and all functional and non-functional testing against an upgraded copy of production.
2. During the configuration upgrade, preserve `MatchSetKey` column data if you want to keep it. This only applies to upgrades from 7.0 versions prior to 7.0.6.
3. Confirm that ContactManager 9.0 Studio shows no compilation errors, and ideally no warnings.
4. Confirm that all components of the upgraded infrastructure will be on a supported release, per the Platform Support Matrix.
5. In production, validate the database schema using the `system_tools -verifydbschema`. Ideally, there will be no issues reported.

6. In production, check the database consistency from the **Server Tools** → **Info Pages** → **Consistency Checks** page. Resolve any errors that are reported.
7. In production, generate a data distribution report. Confirm that the report can be generated properly.
8. In production, if database statistics are not already current, update database statistics shortly before the upgrade deployment.
9. For SQL Server: Decide whether to enable migration to 64-bit IDs during or after the upgrade.
10. For SQL Server: Confirm that the collation setting in the database matches the setting defined in `collations.xml` in the upgraded code base.
11. Prepare all infrastructure, including the database and application servers, load balancer, and so forth.
12. Preserve upgrade instrumentation from the most recent upgrade, if you want to keep it. This information can be downloaded from the **Upgrade Info** page.

Contact Manager Deployment Steps

Perform steps 1-11 in the current production environment.

1. Confirm all batch processing has completed. Ensure no batch process or work queue has a **Status** of **Active** on the **Server Tools** > **Batch Process Info** page.
2. Suspend all message destinations from the **Administration** > **Event Messages** page.
3. Purge completed inactive messages. Note: You can also purge some in advance of the deployment window.
4. Purge completed workflows. Note: You can also purge some in advance of deployment window.
5. Purge completed workflow logs. Note: You can also purge some in advance of deployment window.
6. Validate the database schema using the `system_tools -verifydbschema` command. Ideally, there will be no issues reported. Drop unused columns if necessary, either before or after the upgrade.
7. Check the database consistency from the **Server Tools** → **Info Pages** → **Consistency Checks** page. Ensure no new errors are reported.
8. Generate a data distribution report.
9. Shut down production application servers.
10. Create a database backup.
11. Confirm adequate disk space for the database during upgrade. Allot at least 150% of the current production database size.
12. Disable database replication.
13. For Oracle: Assign default tablespace.
14. For Oracle: Optionally disable logging, statistics update, and statistics update for tables with locked statistics.
15. For SQL Server: Optionally disable SQL Server logging.
16. Move the database from old RDBMS release to new RDBMS release. This step applies to major version upgrades only.
17. For SQL Server: For major version upgrades, set the compatibility level to 110 with the command:
    ```
    ALTER DATABASE dbname SET COMPATIBILITY_LEVEL = 110
    ```
18. Upgrade application servers or repoint the production URL to new application servers as appropriate. This step applies to major version upgrades only.
19. Disable the scheduler in `config.xml`.

20. Enable the database upgrade in `database-config.xml`.
21. Deploy WAR with upgraded configuration to application servers.
22. Start the server to begin database upgrade.
23. Review server log for unexpected errors. Search for the string `ERROR` in the log.
24. Perform initial high-level, view-only testing to ensure system availability.
25. Validate the database schema using the `system_tools -verifydbschema` command. Ideally, there will be no issues reported.
26. Check the database consistency from the **Server Tools** → **Info Pages** → **Consistency Checks** page.Compare the results with the results from the prerequisite steps.
27. Generate a data distribution report. Compare it against the data distribution report generated duringstep 8.
28. Execute custom data validation scripts.
29. Run the Deferred Upgrade Tasks batch process if needed.
30. Run Phone Number Normalizer. Run Phone Number Normalizer on ContactManager first. This step applies to major version upgrades only.
31. Stop the server.
32. Create a database backup.
33. Enable the scheduler in `config.xml`.
34. Disable the database upgrade in `database-config.xml`.
35. Deploy new WAR.
36. Start the server. The upgrade is now completed.
37. Update database statistics by generating the statistics updating statements and executing them.
38. Perform initial testing of key application functionality.
39. Send announcement to user community.
40. For SQL Server: Migrate to 64-bit IDs if not done as part of initial upgrade.

ClaimCenter Prerequisite steps

1. Complete the entire configuration upgrade (code merge) and all functional and non-functional testing against an upgraded copy of production.
2. Confirm that ClaimCenter 9.0 Studio shows no compilation errors, and ideally no warnings.
3. Confirm that all components of the upgraded infrastructure will be on a supported release, per the Platform Support Matrix.
4. Confirm that user workstations will have a supported browser and, if appropriate, be able to work with documents.
5. If on ClaimCenter 6.0 or older, purge orphaned policies if they are consuming a lot of database space.
6. In production, purge address correction records if needed by truncating the `cc_AddressCorrection` table.
7. In production, validate the database schema using the `system_tools -verifydbschema`. Ideally, there will be no issues reported.
8. In production, check the database consistency from the **Server Tools** → **Info Pages** → **Consistency Checks** page. Resolve any errors that are reported.

9. In production, generate a data distribution report. Confirm that the report can be generated properly.
10. In production, if database statistics are not already current, update database statistics shortly before the upgrade deployment.
11. For SQL Server: Decide whether to enable migration to 64-bit IDs during or after the upgrade.
12. For SQL Server: Confirm that the collation setting in the database matches the setting defined in `collations.xml` in the upgraded code base.
13. Prepare all infrastructure, including the database and application servers, load balancer, and so forth.
14. Preserve upgrade instrumentation from the most recent upgrade, if you want to keep it. This information can be downloaded from the **Upgrade Info** page.

ClaimCenter Deployment Steps

Perform steps 1-13 in the current production environment.

1. Confirm all batch processing has completed. Ensure no batch process or work queue has a **Status** of **Active** on the **Server Tools** > **Batch Process Info** page.
2. Suspend all message destinations from the **Administration** > **Event Messages** page.
3. If using ContactManager, formerly known as ContactCenter, run the Contact Auto Sync and Purge Failed Work Items work queues.
4. Purge completed inactive messages. Note: You can also purge some in advance of the deployment window.
5. Purge address correction records if needed by truncating the `cc_AddressCorrection` table.
6. Purge completed workflows. Note: You can also purge some in advance of deployment window.
7. Purge completed workflow logs. Note: You can also purge some in advance of deployment window.
8. If using aggregate limits and if `MigrateToLargeIDsAndDatetime2=true`, truncate the `cc_TmpAggLimitRpt` table.
9. If using catastrophe heat maps, a ClaimCenter 7.0 feature, drop materialized views for catastrophe heat maps if they exist.
10. Validate the database schema using the `system_tools -verifydbschema` command. Ideally, there will be no issues reported. Drop unused columns if necessary, either before or after the upgrade.
11. Check the database consistency from the **Server Tools** → **Info Pages** → **Consistency Checks** page. Ensure no new errors are reported.
12. Generate a data distribution report.
13. Shut down production application servers.
14. Create a database backup.
15. Confirm adequate disk space for the database during upgrade. Allot at least 150% of the current production database size.
16. Disable database replication.
17. For Oracle: Assign default tablespace.
18. For Oracle: Optionally disable logging, statistics update, and statistics update for tables with locked statistics.
19. For SQL Server: Optionally disable SQL Server logging.
20. Move the database from old RDBMS release to new RDBMS release. This step applies to major version upgrades only.

21. For SQL Server: For major version upgrades, set the compatibility level to 110 with the command:
    ```
    ALTER DATABASE dbname SET COMPATIBILITY_LEVEL = 110
    ```
22. Upgrade application servers or repoint the production URL to new application servers as appropriate. This step applies to major version upgrades only.
23. Disable the scheduler in `config.xml`.
24. Enable the database upgrade in `database-config.xml`.
25. Deploy WAR with upgraded configuration to application servers.
26. Start the server to begin database upgrade.
27. Review server log for unexpected errors. Search for the string `ERROR` in the log.
28. Perform initial high-level, view-only testing to ensure system availability.
29. Validate the database schema using the `system_tools -verifydbschema` command. Ideally, there will be no issues reported.
30. Check the database consistency from the **Server Tools** → **Info Pages** → **Consistency Checks** page.Compare the results with the results from the prerequisite steps.
31. Generate a data distribution report. Compare it against the data distribution report generated duringstep 12.
32. Execute custom data validation scripts.
33. Run the Deferred Upgrade Tasks batch process if needed.
34. Run batch processing jobs while at the maintenance run level:
    - Dashboard Statistics
    - Financials Calculations
    - Retired Policy Graph Disconnector
    - Aggregate Limit Calculations (with the `-forceall` option)
35. Run Phone Number Normalizer. Run Phone Number Normalizer on ContactManager first. This step applies to major version upgrades only.
36. Stop the server.
37. Create a database backup.
38. Enable the scheduler in `config.xml`.
39. Disable the database upgrade in `database-config.xml`.
40. Deploy new WAR.
41. Start the server. The upgrade is now completed.
42. Update database statistics by generating the statistics updating statements and executing them.
43. Perform initial testing of key application functionality.
44. Send announcement to user community.
45. For SQL Server: Migrate to 64-bit IDs if not done as part of initial upgrade.

Rolling Upgrade

See also

- “Rolling Upgrade Overview” in the *System Administration Guide*

- "Performing a Rolling Upgrade" in the *System Administration Guide*

Guidewire provides two different ways to modify the configuration of a ClaimCenter server in a cluster:

- Full upgrade
- Rolling upgrade

Each of these upgrade types has associated advantages and disadvantages. You use each upgrade type in different circumstances.

Full Application and Database Upgrade

A full application upgrade has the following functionality:

- Supports either a single or multiple server configuration and upgrade.
- Supports making significant changes to both the ClaimCenter application and database.
- Requires that you stop the entire ClaimCenter production installation for a period of time.
- Requires a significant amount of testing before and after complex changes.

It is possible to start a full upgrade from any server instance in the ClaimCenter cluster, from the Server Tools **Upgrade and Versions** screen.

Rolling Cluster Member Upgrade

A rolling upgrade of the individual server members in the cluster has the following functionality:

- Supports configuration deployment to multiple server instances.
- Successively targets a single cluster member for configuration deployment while leaving the remaining cluster members available for the processing of ClaimCenter operations.
- Supports a limited set of configuration changes.
- Requires that the new target configuration be compatible with the existing source configuration.
- Supports a command-line utility that checks the compatibility of the source and target configurations before deployment.
- Supports configuration deployment management and tracking from within ClaimCenter.

It is possible to start a rolling upgrade from any server instance in the ClaimCenter cluster, from the Server Tools **Upgrade and Versions** screen.

chapter 2

Configuration Upgrade Tools

This topic describes how to obtain and use the Configuration Upgrade Tools.

This topic includes:

Overview of ClaimCenter Configuration Upgrade

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 configuration upgrade process involves four configurations. This guide defines and refers to these configurations as customer, target, old base, and new base.

Customer – The configuration you are now using and will upgrade. This is the ClaimCenter version that you currently run are upgrading from, with your custom configuration applied.

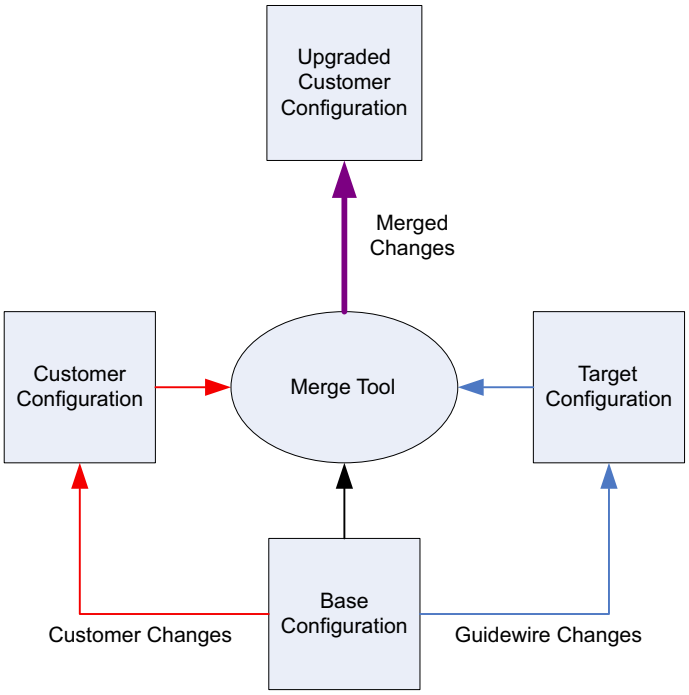
Target – The unedited, Guidewire base configuration of ClaimCenter 9.0. Do not make any modifications to the target configuration prior to completing the configuration upgrade. Do not start Guidewire Studio in the target configuration until you have completed the configuration upgrade.

Old base – The unedited, original, base configuration of the ClaimCenter version you used to create the customer configuration.

New base – The unedited, ClaimCenter 9.0 base configuration.

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.



Configuration Upgrade Tools Introduction

The configuration upgrade tools distribution package contains several tools:

Pre-Upgrade Tool – Runs upgrade steps in the customer configuration (the older version you are upgrading from). For example, if you are upgrading from ClaimCenter 8.0 to ClaimCenter 9.0, the pre-upgrade tool will modify the ClaimCenter 8.0 configuration.

Upgrade Tool – Performs a series of automated upgrade steps.

Merge Tracker Tool – Assists with the manual part of the configuration upgrade process.

Smart Merge Tool – Provides a three-way merge for Guidewire configuration files. As the name suggests, this tool understands the semantics of Guidewire configuration files.

Installing the Configuration Upgrade Tools

The Configuration Upgrade Tools are not included in your ClaimCenter installation. Contact Guidewire Customer Support to obtain the latest version of these tools.

1. Download the latest version of the Guidewire Configuration Upgrade Tools from the Guidewire Resource Portal.
2. Unzip the downloaded ZIP file to your ClaimCenter 9.0 installation directory.
3. Verify that the resulting files and directories are now in the ClaimCenter/upgrade directory.

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`.
2. Click **Project Center** → **Upgrade Services**.
3. Click **Upgrade Diff Reports**.
4. Click **Review Upgrade Diff Reports**.
5. Click **Upgrade Diff Reports - ClaimCenter** or **Upgrade Diff Reports - ContactManager**.
6. Click **Upgrade From** *`base version`*.
7. Click **Upgrade To 9.0**.

Specifying Configuration and Tool Locations

The Configuration Upgrade Tool needs the location of the product that you are upgrading. 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/upgrade/etc/upgrade.properties` file of the target ClaimCenter 9.0 environment. Remove the pound sign, '#', preceding each property to uncomment the line and then specify values. 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 of the top-level ClaimCenter directory of the customer configuration to be upgraded. This directory contains `/studio` and other ClaimCenter directories. Example: `upgrader.priorversion.dir = C:/Guidewire/PolicyCenter-8.0.6`
`upgrader.sourceant.home`	Path to the Ant installation associated with your source customer configuration (the older version you are upgrading from). Example: `upgrader.sourceant.home = C:/Tools/Apache/ant/apache-ant-1.8.2`
`upgrader.sourcejava.home`	Path to the Java JDK installation associated with your source customer configuration (the older version you are upgrading from). Example: `upgrader.sourcejava.home = C:/Program Files/Java/jdk1.7.0_79`

About the Configuration Upgrade Tools

To upgrade your configuration, you must merge Guidewire changes to the base configuration with your changes. Guidewire configuration upgrade tools facilitate this process. Use these tools compare your configuration with the target ClaimCenter version.

The configuration upgrade tools (pre-upgrade and upgrade) perform a series of automated steps. Then you can edit and merge files using the Merge Tracker and the Smart Merge tools in Guidewire Studio.

Guidewire can provide guidance on using the configuration upgrade tools in a multi-user environment using a source control management system.

See the *Upgrade Diffs Report* for an inventory of the differences between the base release and the target release. To retrieve the *Upgrade Diffs Report* follow the procedure described in "Viewing Differences Between Base and Target Releases" on page 27.

Also 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 Pre-upgrade steps

The pre-upgrade steps modify the files of the base environment in place. The base environment is specified by the `upgrader.priorversion.dir` property in `ClaimCenter/upgrade/etc/upgrade.properties` in the *target* environment.

1. Open a command window and set up a ClaimCenter 9.0 development environment.
2. Navigate to the `upgrade\bin` directory of the *target* ClaimCenter 9.0 configuration.
3. Run the following command:
   ```
   preupgrade.bat > preupgrade.log
   ```
4. Verify that the pre-upgrade steps ran to completion.

   The pre-upgrade tool will create the following files.

   In the ClaimCenter customer version:
   - `ClaimCenter/tmp/src-upgrade/server.log`

     (The server is shut down before it completes start-up. This is expected behavior, not an error)
   - `ClaimCenter/tmp/src-upgrade/upgrade-hints.xml` (optional)

   In the ClaimCenter target version:
   - `ClaimCenter/upgrade/bin/preupgrade.log`

Running Upgrade steps

The upgrade steps first copy 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 `ClaimCenter/upgrade/etc/upgrade.properties` in the target environment.

This tool then performs a number of automated steps to upgrade your configuration (in the `tmp/cfg-upgrade/modules` directory) to the target release.

1. Open a command window and set up a ClaimCenter 9.0 development environment.
2. Navigate to the `upgrade\bin` directory of the target ClaimCenter 9.0 configuration.
3. Run the following command:
   ```
   upgrade.bat > upgrade.log
   ```
4. Check the upgrade log for errors.

part II

Upgrading from 8.0

This part describes how to perform an upgrade from ClaimCenter 8.0 to 9.0.

If you are upgrading from ClaimCenter 7.0.x, see “Upgrading from 7.0” on page 53 instead.

This part includes the following topics:

chapter 3

Upgrading the ClaimCenter 8.0 Configuration

This topic describes how to upgrade the ClaimCenter and ContactManager configuration from version 8.0 to 9.0.

If you are upgrading from a 7.0 version, see “Upgrading the ClaimCenter 7.0 Configuration” on page 55 instead.

This topic includes:

Preparing for an Upgrade

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, your entire pre-upgrade ClaimCenter configuration folder (`modules/configuration`) from your custom configuration. A labeled version is a named collection of file revisions.

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

Removing Language Packs

If you have language packs installed, you must remove the language packs before upgrading ClaimCenter. See "Upgrading Display Languages" in the *Globalization Guide*.

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 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 = 110
```

About the Configuration Upgrade Tools

To upgrade your configuration, you must merge Guidewire changes to the base configuration with your changes. Guidewire configuration upgrade tools facilitate this process. Use these tools compare your configuration with the target ClaimCenter version.

The configuration upgrade tools (pre-upgrade and upgrade) perform a series of automated steps. Then you can edit and merge files using the Merge Tracker and the Smart Merge tools in Guidewire Studio.

Guidewire can provide guidance on using the configuration upgrade tools in a multi-user environment using a source control management system.

See the *Upgrade Diffs Report* for an inventory of the differences between the base release and the target release. To retrieve the *Upgrade Diffs Report* follow the procedure described in "Viewing Differences Between Base and Target Releases" on page 27.

Also 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 Pre-upgrade steps

The pre-upgrade steps modify the files of the base environment in place. The base environment is specified by the `upgrader.priorversion.dir` property in `ClaimCenter/upgrade/etc/upgrade.properties` in the *target* environment.

1. Open a command window and set up a ClaimCenter 9.0 development environment.
2. Navigate to the `upgrade\bin` directory of the *target* ClaimCenter 9.0 configuration.
3. Run the following command:
   ```
   preupgrade.bat > preupgrade.log
   ```
4. Verify that the pre-upgrade steps ran to completion.

   The pre-upgrade tool will create the following files.

   In the ClaimCenter customer version:

   - `ClaimCenter/tmp/src-upgrade/server.log`

     (The server is shut down before it completes start-up. This is expected behavior, not an error)
   - `ClaimCenter/tmp/src-upgrade/upgrade-hints.xml` (optional)

   In the ClaimCenter target version:

   - `ClaimCenter/upgrade/bin/preupgrade.log`

Running Upgrade steps

The upgrade steps first copy 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 `ClaimCenter/upgrade/etc/upgrade.properties` in the target environment.

This tool then performs a number of automated steps to upgrade your configuration (in the `tmp/cfg-upgrade/modules` directory) to the target release.

1. Open a command window and set up a ClaimCenter 9.0 development environment.
2. Navigate to the `upgrade\bin` directory of the target ClaimCenter 9.0 configuration.
3. Run the following command:
   ```
   upgrade.bat > upgrade.log
   ```
4. Check the upgrade log for errors.

Configuration Upgrade Automated Steps

The upgrade steps prepares your configuration for the manual 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.

Copying Custom Rules and Adding ClaimCenter 9.0 Default Rules

The upgrade copies customized rules to the target configuration `modules/configuration/config/rules` directory.

This step also copies the default rules provided with ClaimCenter 9.0 to a ClaimCenter 9.0 folder within the `modules/configuration/config/rules` directory of the target configuration. This is so you have a copy of the default rules in a folder in Studio that you can use to compare with your custom rules.

Using the Configuration Upgrade Merge Tools

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 *old customer* file, which you are upgrading.
- The *old base* file, from which you configured the customer file.
- The *target* file, in ClaimCenter 9.0.

In the manual part of the upgrade, you decide whether to use one of these versions unchanged, or merge versions together. The Configuration Upgrade Merge Tracker and Smart Merge tools assist with the manual process.

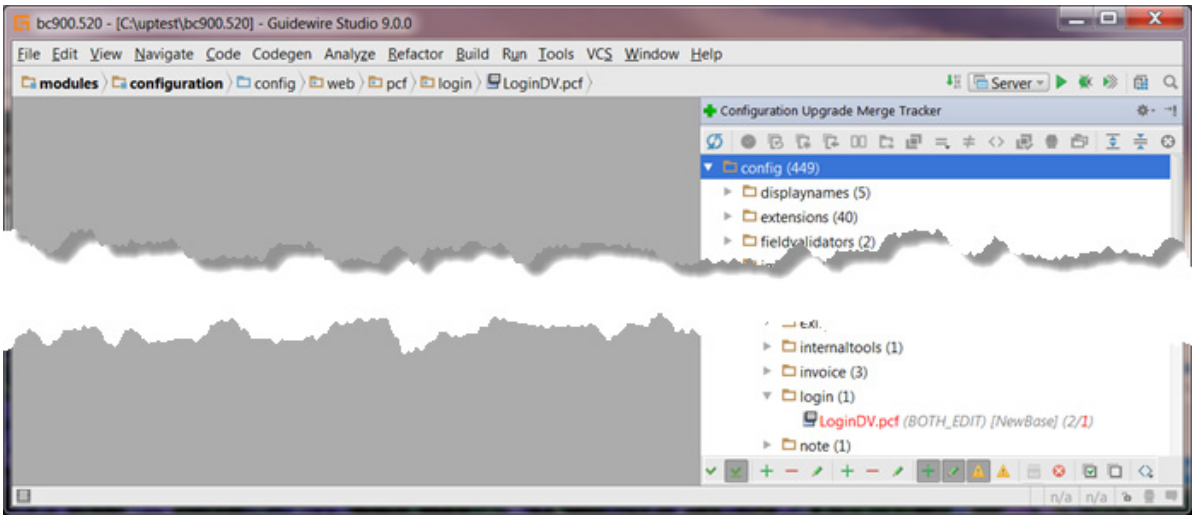
Configuration Upgrade Merge Tracker

The Merge Tracker interface has several important functions:

- It shows a complete list of all configuration files.
- 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 38.
- It allows you to launch the Smart Merge tool.
- It lets you accept the merged version created in the Smart Merge tool.

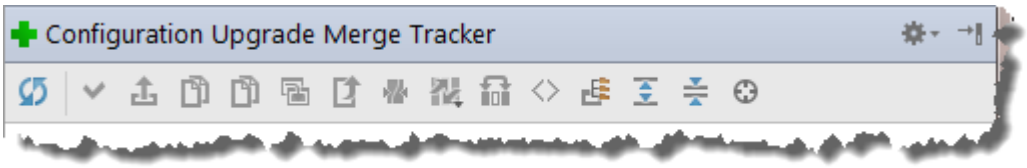
After you have accepted or merged all files that the Merge Tracker displays, the merging process is complete.

The Configuration Upgrade Merge Tracker is a Guidewire Studio plugin. The Merge Tracker displays a tree view of the files in the configuration, filtered by choices selected in the filter bar at the bottom of the tool.



Merge Tracker Toolbar

The top of the Merge Tracker contains the toolbar.



This toolbar includes the following controls:

- **Refresh Button**
- **Compare Button**
- **Export directory tree Button**
- **Merge file Button**
- **Edit tags Button**
- **Merge and resolve files automatically Button**
- **Expand all Button**
- **Collapse all Button**
- **Scroll from source Button**
- **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.

Merge file Button

The Merge file button launches the Smart Merge tool, or if you have configured an alternate merge tool, it will open the tool you have configured in `upgrader.editor.tool` in `upgrade.properties`.

Edit tags Button

The Edit tags button allows you to add or edit *tags* (text strings separated by spaces) for selected items in the file tree. After applying tags, you can show or hide tagged files using the tag filter bar options.

See “Tag Filters” on page 37.

Bulk Action Buttons

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

- **Copy/Mark Resolved**
- **Mark Unresolved**
- **Resolve file**
- **Unresolve and revert to base file**
- **Copy prior customized file**
- **Revert file to base**
- **Mark file as renamed**
- **Unmark file as renamed**

The **Copy prior customized file** button copies the prior customized version of the file to the target configuration. 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** filter

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.

The **Copy/Mark Resolved** button copies files matching the **CUSTOMER_ADD** and **CUSTOMER_EDIT** filters to the target configuration. If there is already a version of a file in the target configuration, then the tool does not copy the file.

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 9.0.

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.

Merge Tracker Filter Bar

The bottom of the Merge Tracker contains the filter bar.

The filter bar includes the following controls:

1. **Resolved Status Filters**
2. **Guidewire Change Status Filters**
3. **Customer Change Status Filters**
4. **Both Change Status Filters**
5. **Unchanged and Unmergeable Status Filters**
6. **Select or Deselect All Filters Buttons**

7. Tag Filters

Resolved Status Filters

The resolved status filter buttons toggle the visibility of resolved and unresolved files. Use these buttons to specify which types of files you want visible in the file tree. For example, you can select **Unresolved** to see only files that are not yet resolved, or you can select **Unresolved** and **Resolved** to show all files.

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 total number of files show after all filters are applied.

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

Guidewire Change Status Filters

These filters are:

- GW_ADD
- GW_DELETE
- GW_EDIT

See "Change Status Filters" on page 38

Customer Change Status Filters

These filters are:

- CUSTOMER_ADD
- CUSTOMER_DELETE
- CUSTOMER_EDIT

See "Change Status Filters" on page 38

Both Change Status Filters

These filters are:

- BOTH_ADD
- BOTH_DELETE
- EDIT_DELETE

See "Change Status Filters" on page 38

Unchanged and Unmergeable Status Filters

These filters are:

- NO_CHANGE
- UNMERGEABLE

See "Change Status Filters" on page 38

Tag Filters

You can add *tags* (text strings separated by spaces) to files using the toolbar. The tag filter options allow you to show or hide tagged files.

See "Edit tags Button" on page 36.

Change Status Filters

This table lists the change status filters that the Configuration Upgrade Merge Tracker displays in the tree view. Use the filter buttons 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	If you resolve the file, the Merge Tracker takes no action. The file, in the new format, already exists in the target configuration. The Merge Tracker automatically marks certain files as unmergeable, including rules files. You can also specify a regular expression pattern in `upgrade.properties` for file paths to mark files matching that pattern as unmergeable. Set the pattern as the value of the `exclude.pattern` property. Typically, you use `exclude.pattern` to specify source control metadata files. Samples are provided in `upgrade.properties` for CVS and SVN.
GW_ADD	add	none	file in target not in base	If you resolve the file, the Merge Tracker takes no action. The file added by Guidewire already exists in the target configuration.
GW_EDIT	edit	none	file in target differs from base	If you resolve the file, the Merge Tracker takes no action. The file added by Guidewire already exists in the target configuration.
GW_DELETE	delete	none	file in base not in target	If you resolve the file, the Merge Tracker takes no action. The file deleted by Guidewire no longer exists in the target configuration.
CUSTOMER_ADD	none	add	file in customer configuration only	If you resolve the file, the Merge Tracker copies the file to the target configuration if the file has not been copied there already.
CUSTOMER_EDIT	none	edit	file differs between customer and base configurations file unchanged between base and target configurations	If you resolve the file, the Merge Tracker copies the file to the target configuration if the file has not been copied there already.
CUSTOMER_DELETE	none	delete	file exists in the base and target configurations but not in the customer configuration	If you click **Delete** on the toolbar, the Merge Tracker removes the file from the target configuration. If you click **Revert to Base** on the toolbar, the Merge Tracker leaves the file in the target configuration.
BOTH_ADD	add	add	new file with matching name in both target and customer configurations (rare)	You must either merge the two versions of the file or copy your prior version of the file into the target configuration before you can resolve the file. Click
BOTH_EDIT	edit	edit	file changed in both customer and target configurations	You must either merge the two versions of the file or copy your prior version of the file into the target configuration before you can resolve the file.

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	If you resolve the file, the Merge Tracker takes no action. If you are sure you want your customized version, you can click **Copy prior customized** in the Merge Tracker toolbar to move the file to the target configuration. 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. If Guidewire has completely removed the file, review the *ClaimCenter New and Changed Guide*, release notes, and the Upgrade Diff 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. 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 filename. 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.
NO_CHANGE	none	none	file not changed from base configuration in either customer or target configurations	If you resolve the file, the Merge Tracker takes no action. The file already exists in the target configuration.

Configuration File Tree

The main panel displays the configuration file tree. Files are shown once, regardless of the number of configurations in which they exist. The number of files in each directory that match the selected change status and resolved status filters is shown in parentheses.

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.

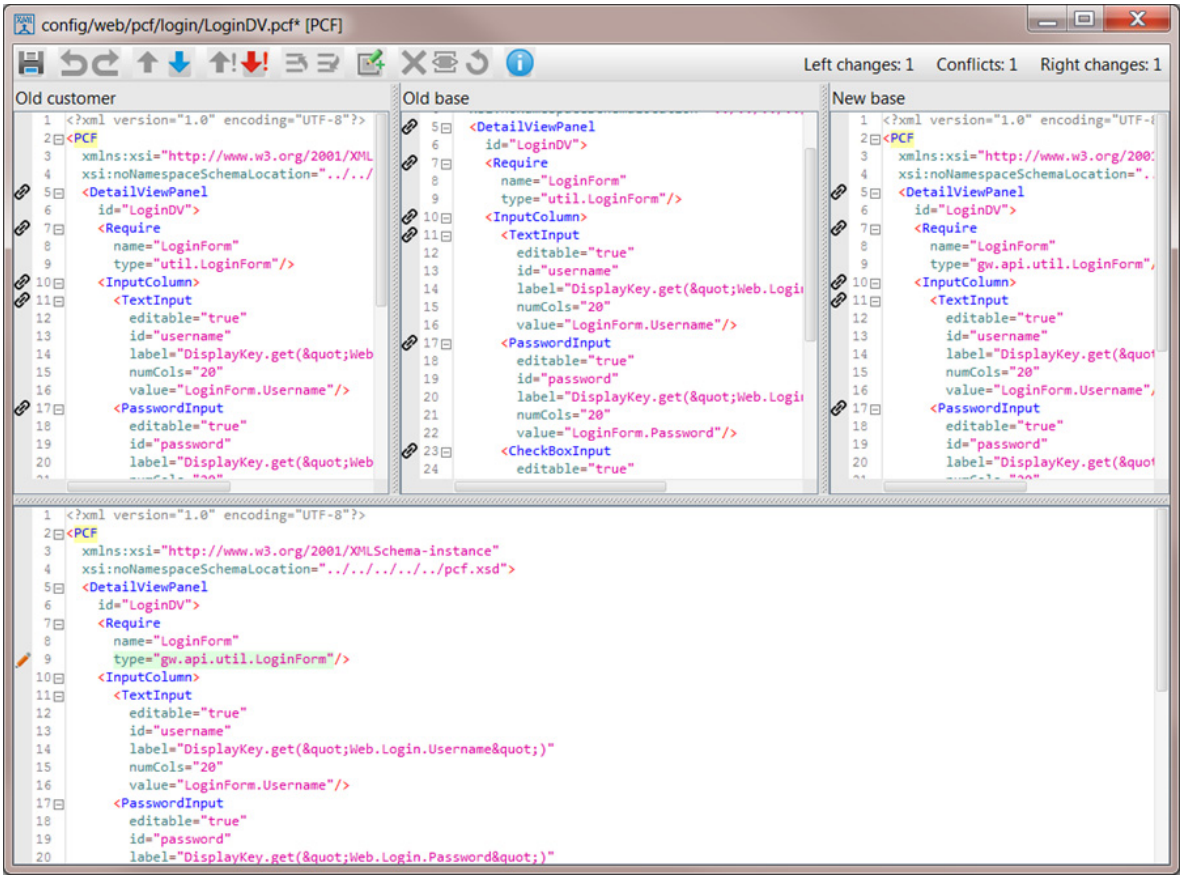
You can use the `pcf.xsd` file in the `modules` directory of the target version to validate merged PCF files.

After you are satisfied with any changes, save the file. The tool always moves files into the target configuration, unless a file is identical to the base or target version. In that case, the tool does not move the file.

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

Configuration Upgrade Smart Merge

The Smart Merge Tool is a three-way merge for Guidewire configuration files. As the name suggests, this tool is not simply an XML merge tool. Smart Merge understands the semantics of Guidewire configuration files, and uses those to assist you in the merge process.



Smart Merge Toolbar

The Smart Merge toolbar

This toolbar includes the following controls:

- **Save**
- **Undo**
- **Redo**
- **Previous Step**

- **Next Step**
- **Previous Conflict**
- **Next Conflict**
- **Move Down**
- **Move Up**
- **Add Comment**
- **Comment Out**
- **Delete Element**
- **Restore**
- **Show Help**

Smart Merge supported file types

The Smart Merge tool supports the following file types:

- Typelist definition files (`.tti`, `.tix`, `.ttx`)
- Data entity metadata files (`.eti`, `.eix`, `.etx`)
- PCF files (`.pcf`)
- Plugin descriptor files (`.gwp`)
- Display name files (`.en`)
- Workflow process files (`.#.xml`)
- Properties files (`.properties`)

Using Smart Merge in standalone mode

A standalone version of Smart Merge tool is available as the `smartmerge` script in the `upgrade/bin` directory.

The `smartmerge` script has three required arguments, *`base`*, *`left`* and *`right`*, corresponding to the three upper panes in the Smart Merge tool. The *`base`* parameter is a base configuration file. The *`left`* parameter is the customer version of the same configuration file. The *`right`* parameter is the new version of the base configuration file.

Configuration Merging Guidelines

The first milestone of an upgrade project is to generate the Java (by running `gwb genJavaApi`) on the target release. To generate the Java APIs, 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 APIs or start the QuickStart application server.

You can generate the Java APIs 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.

After you have generated the Java APIs, you can begin the work of upgrading integrations.

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

In addition to the typical errors described previously, the server might fail to start due to cyclical graph reference errors. See “Identifying Data Model Issues” in the *Upgrade Guide*.

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

After 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.

See the *Upgrade Diffs Report* for an inventory of differences in typekeys between the base release and the target release. To retrieve the *Upgrade Diffs Report* follow the procedure described in “Viewing Differences Between Base and Target Releases” on page 27.

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.

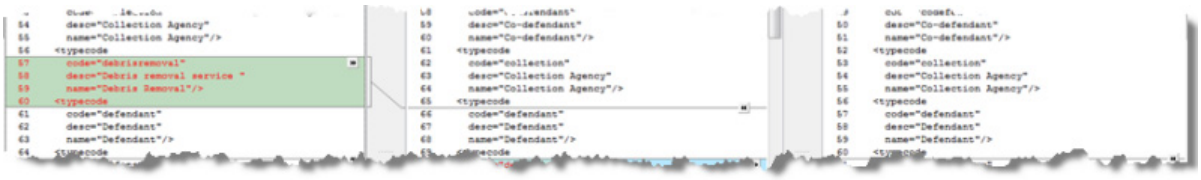
The Configuration Upgrade Tool displays most typelists you have edited in the **CUSTOMER_EDIT** filter. If your edits are simply additional typecodes, accept your version. The automated configuration upgrade may sort a typelist file. If you have also edited the typelist, then it could show the typelist in the **BOTH_EDIT** filter though the only Guidewire change was the sort.

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

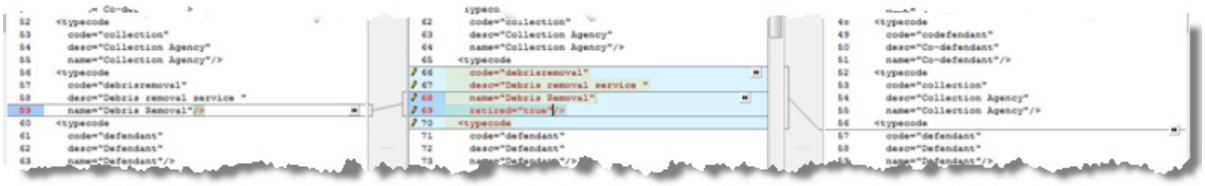
Merging Typelists – Simple Typelists

Merge in new typecodes from the target version, ClaimCenter 9.0. 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.

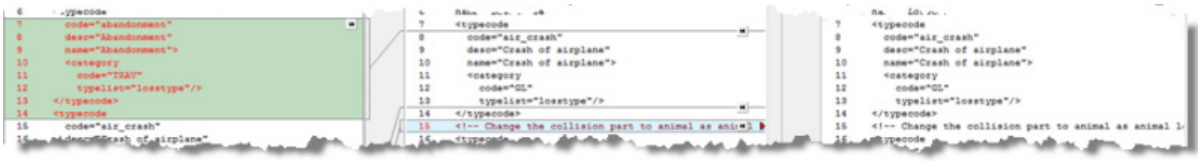


Merging Typelists – Complex Typelists

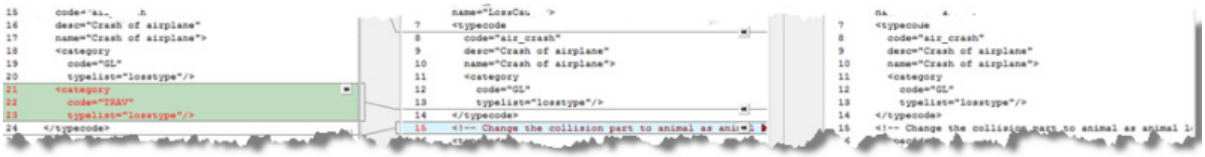
A typecode can reference typecode values from another typelist using the `<category>` subelement. If a new typecode references an existing typecode, do not merge the new typecode unless the referenced typecode is retired. Otherwise, you are defining a new relationship. If the referenced typecode is also 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 `retired` attribute of the typecode to `true`.
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 `retired` attribute of the typecode to `true`.

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.

Reviewing Shared Typekey Configuration

As of version 8.0.3, ClaimCenter enforces restrictions on the use of shared typekeys among subtypes.

Same Field Name and Typelist with Different Column

In ClaimCenter 7.0 and earlier, if a shared typekey had the same field name and typelist, and specified a different column name, ClaimCenter created only one of the typekey columns. The shared typekeys were stored in the single column. As of ClaimCenter 8.0.3, if a shared typekey with the same field name and typelist specifies a different column name, ClaimCenter creates different columns according to the specification. The database upgrade detects shared typekeys using a single column, creates the additional column, and moves the typekey data to the correct column.

Same Field and Column Names with Different Typelists

In ClaimCenter 8.0.1 and 8.0.2, a typekey on subtypes could have the same field name and column name and reference different typelists. As of ClaimCenter 8.0.3, this configuration is not allowed. The database upgrade reports an error if it detects this condition.

If you have subtypes with typekeys with the same field and column name that reference different typelists, update your data model configuration to use different column names for each typelist. The database upgrade then moves data to the new column to match the updated data model.

Merging Lines of Business

Guidewire recommends that you leave LOB typecodes as they were in your prior version. This includes the following typelists:

- `CoverageSubtype`
- `CoverageType`
- `ExposureType`
- `LOBCode`
- `LossType`
- `PolicyType`

This also includes typelists that refer to the LOB typecodes, such as:

- `ClaimTier`
- `CostCategory`
- `CovTermPattern`

- ExposureTier
- LineCategory
- LossPartyType

This also includes other files that refer to the LOB typecodes, such as:

- aggregatelimitused-config.xml
- ISOCoverageMap.csv
- policyperiod-config-xml
- TypeCodeMap.xml
- typecodemapping.xml

If Guidewire has renamed an LOB typecode that you are using, keep the original name. References to the new name will not compile. Update these references to use the original name. Some examples of files where this may need to be done are KeyMetrics.pcf, ClaimInfoBar.pcf, and FNOLWizard.pcf.

Merging Entity Extensions

ClaimCenter 9.0 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.

Reviewing Optional Indexes

Guidewire often adds indexes to entities in the target configuration to improve the performance of database queries in ClaimCenter 9.0. 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 of these indexes if necessary.

Use the Configuration Upgrade Tool to resolve extension 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 ClaimCenter directory of the target version.
2. Run the command gwb genDataDict.

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

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.

Merging Display Properties

The Configuration Upgrade Tool updates display properties files, such as `display.properties` to create a merged file with the extension `.merged`. You could have conflicts in the files if you have a different number of parameters for a key or if you have a different value.

If the number of parameters differs from the prior version, match your parameter set to the new version of the key.

If the value is different, choose which value you want to use in your ClaimCenter configuration.

Merge changes into `display.properties.merged`. When you save the file, the Configuration Upgrade Tool saves it to the configuration module without the `.merged` extension.

If you have added locales, you can export a full list of display keys and typelists from the default ClaimCenter 9.0 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 47.

In ClaimCenter 8.0, Studio trims trailing spaces from display keys by default. You can modify this behavior using the following procedure:

1. Click **File** → **Settings.**
2. Under **IDE Settings** click **Editor.**
3. Under **Other**, change the value of **Strip trailing spaces on Save** to **None.**
4. Click **OK**.

Upgrading Rules to ClaimCenter 9.0

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/configuration/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 to a ClaimCenter 9.0 Rules folder within `modules/configuration/config/rules`. Use Studio to update your rules. You can use the rules in the ClaimCenter 9.0 folder as a comparison. Compare your custom rules to the new default 9.0 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 9.0 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 9.0 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 `ClaimCenter` directory of your starting version.
4. Enter the following command:

   ```
   gwb genRuleReport
   ```

   This command creates a rule repository report XML file in `build/rules`.
5. Append the starting version number to the XML file name.
6. Restore moved directories to the starting version.
7. Install files for a fresh ClaimCenter 9.0 version. This is a separate configuration from the target configuration that you have merged. This version will only contain the default rules provided with ClaimCenter 9.0.
8. Navigate to the `ClaimCenter` directory of the new ClaimCenter 9.0 version.
9. Enter the following command:

   ```
   gwb genRuleReport
   ```

   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. 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.

In Studio, you can compare custom rules to default ClaimCenter 9.0 rules by opening the default rules in the ClaimCenter 9.0 directory within **configuration** → **config** → **Rule Sets**. When you have finished updating all of your custom rules, delete the ClaimCenter 9.0 rules directory from `modules/configuration/config/rules`.

Translating New Display Properties and Typecodes

ClaimCenter 9.0 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. Export display keys by running the following command from your ClaimCenter 9.0 environment `ClaimCenter` directory:

   ```
   gwb exportL10ns -Dexport.file="translation_file" -Dexport.locale="language to export"
   ```

2. Open the exported translation 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.
3. Specify localized values for the untranslated properties.
4. Save the updated file.
5. Import the updated file by running the following command from your ClaimCenter 9.0 environment `ClaimCenter` directory:

   ```
   gwb importL10ns -Dimport.file="translation_file" -Dimport.locale="language to import"
   ```

   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 ContactManager), 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 `gwb studio` command in the `ClaimCenter` directory.
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.

Web Services for Claims

In ClaimCenter, the `Check` entity now contains `MailingAddress`, a foreign key reference to the `Address` entity, instead of the deprecated `MailToAddress` field. In order to accommodate this change, previous versions of web services, `ClaimFinancialsAPI` and `ClaimAPI`, have been removed from the application.

Validating the ClaimCenter 9.0 Configuration

This topic includes procedures to validate the upgraded configuration.

Using Studio to Verify Files

You can use Studio to verify classes and enhancements, including libraries, PCF files, rules, and typelists without having to start ClaimCenter. Do not start ClaimCenter at this point. Studio can run without connecting to the application server.

To validate Studio resources

1. Start Guidewire Studio by running `gwb studio` from the `ClaimCenter` directory.
2. Click **Analyze** → **Inspect Code...**.
3. Set the **Inspection scope** to **module 'configuration'**.
4. Click OK. Studio runs inspections to identify incorrect Gosu syntax, issuing either a warning or an error.
5. Correct all identified errors with Studio. You can defer fixing warnings.

Starting ClaimCenter and Resolving Errors

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" in the *Installation Guide* and "Deploying ClaimCenter to the Application Server" 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, such as 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.

ClusteringComponent Class Removed

The `ClusteringComponent` class has been removed.

Replace usages such as:

```
ServerDependencies.getClusteringComponent().isClusteringEnabled()
```

With:

```
ClusterEnabledInfo.isClusteringEnabled().
```

Building and Deploying ClaimCenter 9.0

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” 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, then upgrade, build, and deploy each individually before attempting to reintegrate 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` 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. Building and deploying a WAR or EAR file copies the JAR file into the appropriate place for it to be accessed by ClaimCenter. JAR files in this location survive the upgrade process.

chapter 4

Upgrading the ClaimCenter 8.0 Database

To upgrade your ClaimCenter Database, refer to “Upgrading the ClaimCenter 8.0 Database” in the *ClaimCenter Upgrade Guide*, which is included with the main ClaimCenter documentation set.

part III

Upgrading from 7.0

This part describes how to perform an upgrade from ClaimCenter 7.0 to 9.0.

If you are upgrading from ClaimCenter 8.0, see “Upgrading from 8.0” on page 29 instead.

This part includes the following topics:

chapter 5

Upgrading the ClaimCenter 7.0 Configuration

This topic describes how to upgrade the ClaimCenter and ContactManager configuration from version 7.0 to 9.0.

If you are upgrading from an 8.0 version, see “Upgrading the ClaimCenter 8.0 Configuration” on page 31 instead.

This topic includes:

Preparing for an Upgrade

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, your entire pre-upgrade ClaimCenter configuration folder (`modules/configuration`) from your custom configuration. A labeled version is a named collection of file revisions.

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

Removing Language Packs

If you have language packs installed, you must remove the language packs before upgrading ClaimCenter. See “Upgrading Display Languages” in the *Globalization Guide*.

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 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 = 110
```

About the Configuration Upgrade Tools

To upgrade your configuration, you must merge Guidewire changes to the base configuration with your changes. Guidewire configuration upgrade tools facilitate this process. Use these tools compare your configuration with the target ClaimCenter version.

The configuration upgrade tools (pre-upgrade and upgrade) perform a series of automated steps. Then you can edit and merge files using the Merge Tracker and the Smart Merge tools in Guidewire Studio.

Guidewire can provide guidance on using the configuration upgrade tools in a multi-user environment using a source control management system.

See the *Upgrade Diffs Report* for an inventory of the differences between the base release and the target release. To retrieve the *Upgrade Diffs Report* follow the procedure described in "Viewing Differences Between Base and Target Releases" on page 27.

Also 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 Pre-upgrade steps

The pre-upgrade steps modify the files of the base environment in place. The base environment is specified by the `upgrader.priorversion.dir` property in `ClaimCenter/upgrade/etc/upgrade.properties` in the *target* environment.

1. Open a command window and set up a ClaimCenter 9.0 development environment.
2. Navigate to the `upgrade\bin` directory of the *target* ClaimCenter 9.0 configuration.
3. Run the following command:
   ```
   preupgrade.bat > preupgrade.log
   ```
4. Verify that the pre-upgrade steps ran to completion.

   The pre-upgrade tool will create the following files.

   In the ClaimCenter customer version:
   - `ClaimCenter/tmp/src-upgrade/server.log`

     (The server is shut down before it completes start-up. This is expected behavior, not an error)
   - `ClaimCenter/tmp/src-upgrade/upgrade-hints.xml` (optional)

   In the ClaimCenter target version:
   - `ClaimCenter/upgrade/bin/preupgrade.log`

Running Upgrade steps

The upgrade steps first copy 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 `ClaimCenter/upgrade/etc/upgrade.properties` in the target environment.

This tool then performs a number of automated steps to upgrade your configuration (in the `tmp/cfg-upgrade/modules` directory) to the target release.

1. Open a command window and set up a ClaimCenter 9.0 development environment.
2. Navigate to the `upgrade\bin` directory of the target ClaimCenter 9.0 configuration.
3. Run the following command:
   ```
   upgrade.bat > upgrade.log
   ```
4. Check the upgrade log for errors.

Configuration Upgrade Automated Steps

The upgrade steps prepares your configuration for the manual 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.

Removing Template Pages

The Configuration Upgrade Tool deletes PCF template pages. These pages have a `<TemplatePage>` root element. The upgrade also removes `<EntryPoint>` elements that reference template pages. Template pages have been replaced by SOAP-based data integration in ClaimCenter 8.0.

Updating PCF Files

The Configuration Upgrade Tool performs the following modifications to PCF files:

- Removes the `reflectOnBottom` attribute. This attribute was used to display the a virtual toolbar at the bottom of a page. The attribute was removed because the user interface needs to match the server configuration. No alternative configuration is available.
- Converts all `postOnChange` attributes on a value widget to a child `PostOnChange` node. For example, the upgrade converts:

  ```
  <Input id="xxx" postOnChange="true" onChange="someMethod()" disablePostOnEnter="doEvaluation()"/>
  ```

  to:

  ```
  <Input id="xxx">
    <PostOnChange onChange="someMethod()" disablePostOnEnter="doEvaluation()"/>
  </Input>
  ```
- Removes the `showNoneSelected` attribute from all `DetailView` inputs that are bound to a value. Setting `showNoneSelected=false` would suppress the **None Selected** option from drop-down lists and would default to the first option. This type of configuration was incorrect because the selection of the option was generally programmatically incorrect and was often used as a shortcut instead of specifying an explicit default. Verify all removals to ensure there is not any dependent logic. If there is, specify an explicit default in the page configuration.
- Removes the `showNoneSelected` attribute from all `<ValueCellType>` nodes. See the above note about removal of the `showNoneSelected` attribute from all `DetailView` inputs that are bound to a value.
- Removes the `numDataEntriesPerRow` and `transposed` attributes from `RowIteratorNode` elements. Transposed lists are a relatively rare configuration. If you had one in your configuration, use a traditional list view.
- Removes `<DetailViewPanel>` elements from `<ButtonCell>`, `<ButtonInput>`, and `<ToolbarButtonType>` elements. Detail views can no longer be embedded inside buttons.
- Converts `valueWidth` attributes on cell widgets to `value` attributes. As of 8.0, ClaimCenter sizes cells by heuristics rather than content, so `valueWidth` is no longer necessary.
- If all cells in a row have the `useHeaderStyle="true"` property, the upgrade moves the property to the row level. A list can only have one header. See below.
- Updates rows to rename the `useHeaderStyle` property to `renderAsSmartHeader`. The property is renamed because the header functionality is more than styling. When a row is rendered as a smart header, all the row header interactive features are made available.
- Renames `<ContentCell>` elements to `<Link>`.
- Converts `<Cell>` elements within `<ColumnFooter>` to `<TextCell>` elements.
- Removes any element that is not a `<TextCell>` element from `<ColumnFooter>` elements.
- Removes `<ColumnHeader>` elements from `<CellType>` elements.
- Remove `<DetailViewPanel>` from `<ContentCell>`. The upgrade performs the following steps. After the automatic upgrade, review your `<ContentCell>` configurations to manually verify the configuration and make any changes. Content cells cannot have editable detail views embedded in them. Review all removals to ensure functionality. If editable content is needed within a row of data, the recommended configuration is a list detail panel.

  - For any <ContentCell> that contains a <DetailViewPanel>, the upgrade renames the <ContentCell> to <FormatCell>.
  - For other types of <ContentCell>, the upgrade renames the element to <LinkCell>.
  - Removes elements that are not allowed in the <FormatCell>, such as <DetailViewPanel> and <InputColumn>. This strips out unnecessary container elements. No content will be removed.
  - Renames inputs in the <DetailViewPanel> to <TextInput> unless they are <ContentInput>, <TextInput>, or <NoteBodyInput>.
  - Removes attributes that were allowed on specific input elements but not on <TextInput>.
- Removes the useHeaderStyle attribute from all cells that can be bound to a value. The header style in 8.0 is a lot more extensive. Smart header capabilities have been added, in addition to the styling. Header capabilities are at the row level as opposed to the cell. If you are interested in highlighting content, there are a few other ways to achieve that. Review the PCF reference for a full list of attributes for that particular cell variant.
- Removes the compress attribute from <DetailViewPanel>.
- Removes the compress attribute from <ListViewPanel>.
- Removes the compressIfSingleChild attribute from <InputGroup>.
- Comments out <ProgressCell> elements. This was an uncommon widget that Guidewire has removed. If you were using it on some page and would like to continue to do so, create a list detail panel, and use the ProgressInput in the detail section instead.
- Removes the refreshOnProgressComplete attribute from <ListViewPanel> and <Row> elements. This is part of the removal of the <ProgressCell> widget.
- Removes the following attributes from <ChartPanel>:
  - bgColor
  - border
  - displayPlotOutline
  - orientation
  - sameSeriesColor
  - threeD
  - tooltip

  Guidewire cleaned up the <ChartPanel> schema as a part of simplification and a move to a more interactive experience.
- Removes the following attributes from <DomainAxis>:
  - autoRange
  - autoRangeIncludesZero
  - tickUnit
  - upperMargin
- Removes the <Interval> element.
- Removes the following attributes from <RangeAxis>:
  - autoRange
  - autoRangeIncludesZero
  - tickUnit
  - upperMargin
- Removes the percentComplete attribute from <DataSeries>.
- Removes the following from <DualAxisDataSeries>:
  - autoRangeIncludesZero
  - lowerMargin

  - tickUnit
  - tooltip
  - upperMargin
- Removes the following chart types from the <ChartType> enumerator:
  - Waterfall
  - Gantt
- Renames the following chart types in the <ChartType> enumerator:
  - Dial → Gauge
  - Polar → Radar
  - Ring → Pie
  - StackedArea → Area (there is no more distinction between a stacked vs non-stacked area)
  - XYStep → XYLine
  - XYStepArea → XYArea

Upgrading Work Queue Configuration

The Configuration Upgrade Tool makes the following changes to work-queue.xml:

- removes obsolete minpollinterval attribute.
- removes obsolete orphansFirst attribute.
- removes obsolete checkInAfterError attribute
- adds retryInterval=*value* (the upgrade sets the value to 0 if checkInAfterError was true, or to the current value of progressinterval if checkInAfterError was not true.)

Upgrading Database Configuration

The Configuration Upgrade Tool moves the database configuration from config.xml to database-config.xml and converts it to the ClaimCenter 8.0 format.

As of ClaimCenter 8.0, Guidewire has made the following changes to the database configuration:

- The <database> element no longer contains subelements with the following syntax:

  <param name="*name*" value="*value*">
- For Oracle, the <tablespacemapping> elements have been replaced with a single <tablespaces> element. The <tablespaces> element is contained in an <ora-db-ddl> parent element. The <tablespaces> element includes the attributes admin, index, op, staging, typelist, and lob. These attributes correspond to the logical tablespaces defined in ClaimCenter. You can use these attributes to map tablespaces that you have created to the logical tablespaces.
- For SQL Server, the <tablespacemapping> elements have been replaced with a single <mssql-filegroups> element. The <mssql-filegroups> element is contained in an <mssql-db-ddl> parent element. The <mssql-filegroups> element includes the attributes admin, index, op, staging, typelist, and lob. These attributes correspond to the logical tablespaces defined in ClaimCenter. You can use these attributes to map file groups that you have created to the logical tablespaces.
- If a <tablegroup> element was contained in a <database> element that had an env attribute defined, the upgrade copies the env attribute onto the <tablegroup> element.
- If any of the following <database> attributes are defined, the upgrade copies them over to the <database> element in database-config.xml: addforeignkeys, autoupgrade, checker, dbtype, env, name, printcommands. The schema for these attributes has not changed.
- If any comments exist within the <database> element, the upgrade copies these comments to the <database> element in database-config.xml.

- If the `driver` attribute of the `<database>` element equals `dbcp`, the upgrade adds a `<dbcp-connection-pool>` element and copies the `jdbcUrl` parameter to the `jdbc-url` attribute of the `<dbcp-connection-pool>` element. If the original configuration did not include a `jdbcUrl` parameter, then the upgrade logs an error. If a `passwordFile` attribute is specified on the `<database>` element of the old configuration, the upgrade transfers the `passwordFile` attribute to the `<dbcp-connection-pool>` element. The upgrade converts any of the following parameters defined in the old configuration to attributes on the `<dbcp-connection-pool>` element:
  - `maxActive`
  - `maxIdle`
  - `maxWait`
  - `minEvictableIdleTimeMillis`
  - `numTestsPerEvictionRun`
  - `testOnBorrow`
  - `testOnReturn`
  - `testWhileIdle`
  - `timeBetweenEvictionRunsMillis`
  - `whenExhaustedAction`
- If the `driver` attribute of the `<database>` element equals `dbcp` and any of the following attributes are set, the upgrade creates a `<reset-tool-params>` element within the `<dbcp-connection-pool>` element:
  - `collation`
  - `oracle.tnsnames`
  - `system.username`
  - `system.password`

  The upgrade then transfers any of these attributes that are defined to the new `<reset-tool-params>` element.
- If the `driver` attribute of the `<database>` element equals `jndi`, the upgrade adds a `<jndi-connection-pool>` element and copies the `jndi.datasource.name` parameter to the `datasource-name` attribute of the `<jndi-connection-pool>` element. If the original configuration did not include a `jndi.datasource.name` parameter, then the upgrade logs an error.
- If the old configuration includes an `<upgrade>` element within the `<database>` element, the upgrade adds an `<upgrade>` element to the `<database>` element of the new configuration.
- If the old configuration contains an `<upgrade>` element that includes an `oracleMarkColumnsUnused` attribute, the upgrade converts the attribute to a `deferDropColumns` attribute, preserving the value.
- If the old configuration contains an `<upgrade>` element that includes a `verifySchema` attribute, the upgrade copies this attribute to `<upgrade>` element of the new configuration.
- If the old configuration contains an `<upgrade>` element that contains an `<oracleddloptions>` or `<sqlserverddlopts>` element, the upgrade logs a warning. You must upgrade these elements manually.
- If the old configuration includes a `<databasestatistics>` element within the `<database>` element, the upgrade copies the `<databasestatistics>` element to the `<database>` element of the new configuration.
- For Oracle databases, if the `<database>` element includes any of the following parameters, the upgrade creates an `<oracle-settings>` element within the `<database>` element of the new configuration:
  - `queryRewriteEnabled`
  - `statisticsLevel`
  - `stored.outlines`
  - `UseDbResourceMgrCancelSql`

  The upgrade converts any of the above parameters to attributes on the new `<oracle-settings>` element. The attributes have the following names:
  - `query-rewrite`

  - statistics-level-all (if any value is set for statisticsLevel in the old configuration, the upgrade sets the statistics-level-all attribute to true in the new configuration. The value ALL was the only valid value for the statisticsLevel parameter in the old configuration.)
  - stored-outline-category
  - db-resource-mgr-cancel-sql
- For SQL Server databases, if the <database> element includes either the msjdbctracelevel or msjdbctracefile parameter, the upgrade adds a <sqlserver-settings> element within the <database> element of the new configuration. The upgrade then converts the msjdbctracelevel and msjdbctracefile parameters to jdbc-trace-level and jdbc-trace-file attributes on the <sqlserver-settings> element.
- For SQL Server databases, if the unicodecolumns parameter is defined in the old configuration, the upgrade adds a unicodecolumns attribute to the <sqlserver-settings> element of the new configuration. If the <sqlserver-settings> element has not yet been created, the upgrade creates the element.
- If any <tablespacemapping> elements are defined in the old configuration, the upgrade creates an <upgrade> element within the <database> element of the new configuration if one does not yet exist. The upgrade then does the following, depending on the database type:
  - For Oracle, the upgrade adds an <ora-db-ddl> element within the <upgrade> element of the new configuration, if an <ora-db-ddl> element is not yet defined. The upgrade then adds a <tablespaces> element to the <ora-db-ddl> element and converts each <tablespacemapping> element to an attribute on the <tablespaces> element. The upgrade then adds an <ora-lobs> element to the <ora-db-ddl> element and sets the <ora-lobs> attribute type to BASICFILE. Although Oracle 12c creates SecureFile LOB columns by default, the configuration upgrade sets the default type for any new LOBs to BASICFILE to maintain consistency with the Oracle 11 default. If Oracle LOBs are configured to be SecureFile or compressed SecureFiles, the configuration upgrade does not transfer the DDL settings to database-config.xml. These configuration settings must be applied to the new database-config.xml database element manually. Note that if you change a DDL configuration, the setting only applies for new objects.
  - For SQL Server, the upgrade adds an <mssql-db-ddl> element within the <upgrade> element of the new configuration, if an <mssql-db-ddl> element is not yet defined. The upgrade then adds a <mssql-filegroups> element to the <mssql-db-ddl> element and converts each <tablespacemapping> element to an attribute on the <mssql-filegroups> element.

Splitting Localization.xml into Separate Files for each Locale

The upgrade splits the locales from the single localization.xml file used in ClaimCenter 7.0 into a separate file for each locale defined by a <GWLocale> element. The new location for each split localization.xml file is config/locale/*locale*/. Each localization.xml file can have only one GWLocale element in ClaimCenter 8.0.

Splitting address-config.xml into Separate Files for each Country

The upgrade splits the address format definitions from the single address-config.xml file used in ClaimCenter 7.0 into a separate file for each country defined by an <AddressDef> element. The new location for each split address-config.xml file is config/geodata/*country code*/. Each address-config.xml file can have only one AddressDef element in ClaimCenter 8.0.

Splitting zone-config.xml into Separate Files for each Country

The upgrade splits the zone configuration definitions from the single zone-config.xml file used in ClaimCenter 7.0 into a separate file for each country. Zones for each country are defined by a <Zones> element with a countryCode attribute. The new location for each split zone-config.xml file is config/geodata/*country code*/. In ClaimCenter 8.0 each zone-config.xml file can have only one <Zones> element that contains zones for a single country.

Splitting currencies.xml into Separate Files for each Currency

The upgrade splits the currency definitions from the single `currencies.xml` file used in ClaimCenter 7.0 into a separate file for each currency type. Each currency type is defined by a `<CurrencyType>` element with a `code` attribute. The separate files are each named `currency.xml`. The new location for each `currency.xml` file is `config/currencies/`*`code`*`/`, where *`code`* is the value of the code attribute on the `<CurrencyType>` element.

Moving Country-based Field Validator Definition Files

The upgrade moves each country-based field validator definition file to an individual directory. Country-specific field validator definition files are named with the format `fieldvalidators_`*`country code`*`.xml`, such as `fieldvalidators_JP.xml` for field validators specific to Japan. The upgrade moves each country-specific field validator definition file to `config/fieldvalidators/`*`country code`*`/`. The generic fieldvalidators.xml file remains at `config/fieldvalidators/`.

Moving Rules Files up One Directory

The upgrade moves all rules files up one directory from `config/rules/rules/` to `config/rules/`.

Reformatting Rules for Display in Studio Rules Editor

The upgrade reformats `.gr` rule files so that the Studio rules editor recognizes the file contents as rules.

Copying Custom Rules and Adding ClaimCenter 9.0 Default Rules

The upgrade copies customized rules to the target configuration `modules/configuration/config/rules` directory.

This step also copies the default rules provided with ClaimCenter 9.0 to a ClaimCenter 9.0 folder within the `modules/configuration/config/rules` directory of the target configuration. This is so you have a copy of the default rules in a folder in Studio that you can use to compare with your custom rules.

Renaming SOAP Web Services from XML to RWS

The upgrade changes the extension of SOAP web service files in `config/webservices` from `.xml` to `.rws`.

Renaming Plugins from XML to GWP

The upgrade changes the extension of plugin files in `config/plugin/registry` from `.xml` to `.gwp`.

Renaming Display Names Files from XML to EN

The upgrade changes the extension of display names files in `config/displaynames` from `.xml` to `.en`.

Upgrading Display Keys

The upgrade compares display keys from the custom configuration with display keys in the base 7.0 configuration and display keys in the default ClaimCenter 8.0 configuration. The following display key files are inspected.

- `display.properties`
- `gosu.display.properties`
- `productmodel.display.properties`
- `studio.display.properties`
- `typelist.properties`

The upgrade compares the case of display property keys in the custom configuration with the case of the key in the default ClaimCenter 9.0 configuration. If the case does not match, but the value assigned to the key matches the value in the default configuration, the upgrade corrects the case in the custom configuration. If the case of the keys does not match, and the value is different in the custom configuration, the upgrade reports an error.

The upgrade then merges the display keys files into a single file for each locale. This file has the extension .merged. The merged display properties files are available in the Configuration Upgrade Tool for comparison with the default ClaimCenter `display.properties`. You can merge Guidewire changes and new properties with your custom properties values.

Adding nullok="true" to Entity and Extension Foreign Key Columns

The upgrade modifies ETI and EIX files in `config/metadata` and ETX and ETI files in `config/extensions`. The upgrade adds the attribute `nullok="true"` to `<foreignkey>` and `<edgeForeignKey>` elements if the element did not explicitly specify a value for the `nullok` attribute. In ClaimCenter 8.0, the `nullok` attribute is required to be explicitly set.

Removing deletefk Attribute from Entity and Extension Foreign Keys

The upgrade removes the `deletefk` attribute from all `<foreignkey>` and `<edgeforeignkey>` elements that include a `deletefk` attribute.

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.

The namespace was encouraged but optional prior to 8.0. The namespace must be specified in 8.0.

Upgrading Document Assistant Parameters

In ClaimCenter 8.0, Guidewire Document Assistant uses a Java applet deployed using JNLP instead of an ActiveX control. The upgrade updates `config.xml` for this change. In this step, the upgrade replaces legacy Document Assistant ActiveX configuration parameters with the updated ones. The upgrade makes the following changes:

- Renames `AllowActiveX` to `AllowDocumentAssistant`, ignoring the old value. In 8.0 `AllowDocumentAssistant` defaults to `false`, whereas `AllowActiveX` was `true` in prior releases. The deployment, security, and configuration of applets is entirely different from ActiveX controls. Consider Java security issues as part of your decision to deploy the Document Assistant applet.
- Renames `UseGuidewireActiveXControlToDisplayDocuments` to `UseDocumentAssistantToDisplayDocuments`, keeping the old value.
- Removes `AllowActiveXAutoInstall`.
- Removes `UseDocumentNameAsFileName`.
- Adds `DocumentAssistantJNLP`.

Separating Entities and Typelists

The upgrade creates `entity` and `typelist` folders in `config/metadata` and `config/extensions` directories. The upgrade then moves ETI, EIX, and ETX files into the `entity` folders and moves TTI, TIX, and TTX files into the `typelist` folders.

Upgrading aggregatelimitconfig-used.xml

The `aggregatelimitconfig-used.xml` file has a different format in ClaimCenter 8.0 than in previous versions. If you have configured cost types in your custom configuration, the upgrade converts this file to the ClaimCenter 8.0 format.

Using the Configuration Upgrade Merge Tools

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 *old customer* file, which you are upgrading.
- The *old base* file, from which you configured the customer file.
- The *target* file, in ClaimCenter 9.0.

In the manual part of the upgrade, you decide whether to use one of these versions unchanged, or merge versions together. The Configuration Upgrade Merge Tracker and Smart Merge tools assist with the manual process.

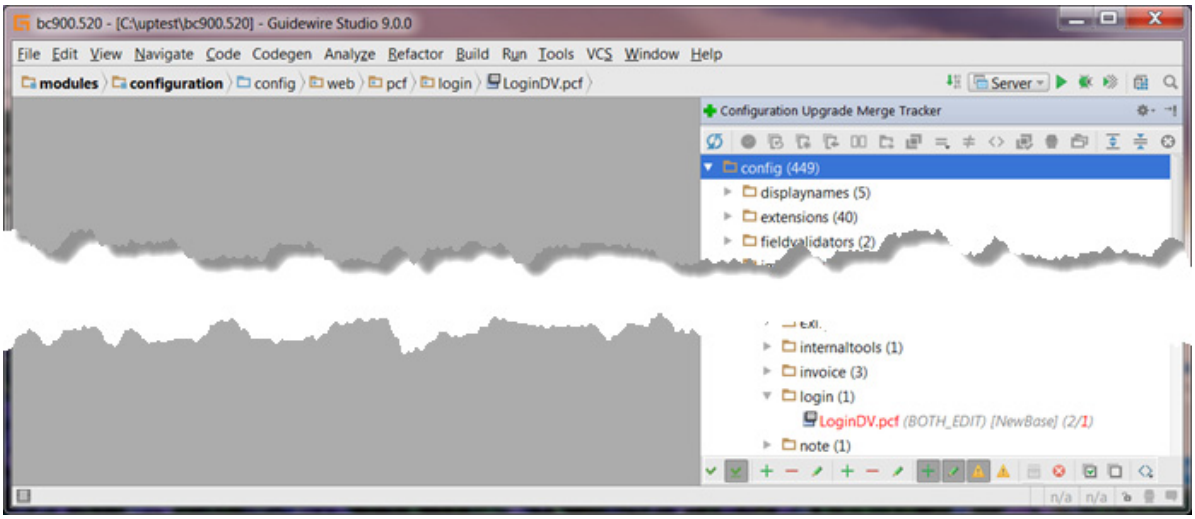
Configuration Upgrade Merge Tracker

The Merge Tracker interface has several important functions:

- It shows a complete list of all configuration files.
- 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 69.
- It allows you to launch the Smart Merge tool.
- It lets you accept the merged version created in the Smart Merge tool.

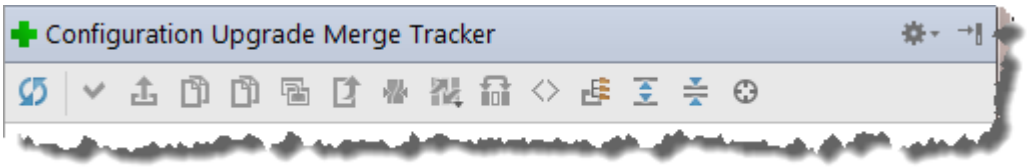
After you have accepted or merged all files that the Merge Tracker displays, the merging process is complete.

The Configuration Upgrade Merge Tracker is a Guidewire Studio plugin. The Merge Tracker displays a tree view of the files in the configuration, filtered by choices selected in the filter bar at the bottom of the tool.



Merge Tracker Toolbar

The top of the Merge Tracker contains the toolbar.



This toolbar includes the following controls:

- **Refresh Button**
- **Compare Button**
- **Export directory tree Button**
- **Merge file Button**
- **Edit tags Button**
- **Merge and resolve files automatically Button**
- **Expand all Button**
- **Collapse all Button**
- **Scroll from source Button**
- **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.

Merge file Button

The Merge file button launches the Smart Merge tool, or if you have configured an alternate merge tool, it will open the tool you have configured in `upgrader.editor.tool` in `upgrade.properties`.

Edit tags Button

The Edit tags button allows you to add or edit *tags* (text strings separated by spaces) for selected items in the file tree. After applying tags, you can show or hide tagged files using the tag filter bar options.

See “Tag Filters” on page 68.

Bulk Action Buttons

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

- **Copy/Mark Resolved**
- **Mark Unresolved**
- **Resolve file**
- **Unresolve and revert to base file**
- **Copy prior customized file**
- **Revert file to base**
- **Mark file as renamed**
- **Unmark file as renamed**

The **Copy prior customized file** button copies the prior customized version of the file to the target configuration. 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** filter

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.

The **Copy/Mark Resolved** button copies files matching the **CUSTOMER_ADD** and **CUSTOMER_EDIT** filters to the target configuration. If there is already a version of a file in the target configuration, then the tool does not copy the file.

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 9.0.

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.

Merge Tracker Filter Bar

The bottom of the Merge Tracker contains the filter bar.

The filter bar includes the following controls:

1. **Resolved Status Filters**
2. **Guidewire Change Status Filters**
3. **Customer Change Status Filters**
4. **Both Change Status Filters**
5. **Unchanged and Unmergeable Status Filters**
6. **Select or Deselect All Filters Buttons**

7. Tag Filters

Resolved Status Filters

The resolved status filter buttons toggle the visibility of resolved and unresolved files. Use these buttons to specify which types of files you want visible in the file tree. For example, you can select **Unresolved** to see only files that are not yet resolved, or you can select **Unresolved** and **Resolved** to show all files.

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 total number of files show after all filters are applied.

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

Guidewire Change Status Filters

These filters are:

- GW_ADD
- GW_DELETE
- GW_EDIT

See "Change Status Filters" on page 69

Customer Change Status Filters

These filters are:

- CUSTOMER_ADD
- CUSTOMER_DELETE
- CUSTOMER_EDIT

See "Change Status Filters" on page 69

Both Change Status Filters

These filters are:

- BOTH_ADD
- BOTH_DELETE
- EDIT_DELETE

See "Change Status Filters" on page 69

Unchanged and Unmergeable Status Filters

These filters are:

- NO_CHANGE
- UNMERGEABLE

See "Change Status Filters" on page 69

Tag Filters

You can add *tags* (text strings separated by spaces) to files using the toolbar. The tag filter options allow you to show or hide tagged files.

See "Edit tags Button" on page 67.

Change Status Filters

This table lists the change status filters that the Configuration Upgrade Merge Tracker displays in the tree view. Use the filter buttons 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	If you resolve the file, the Merge Tracker takes no action. The file, in the new format, already exists in the target configuration. The Merge Tracker automatically marks certain files as unmergeable, including rules files. You can also specify a regular expression pattern in `upgrade.properties` for file paths to mark files matching that pattern as unmergeable. Set the pattern as the value of the `exclude.pattern` property. Typically, you use `exclude.pattern` to specify source control metadata files. Samples are provided in `upgrade.properties` for CVS and SVN.
GW_ADD	add	none	file in target not in base	If you resolve the file, the Merge Tracker takes no action. The file added by Guidewire already exists in the target configuration.
GW_EDIT	edit	none	file in target differs from base	If you resolve the file, the Merge Tracker takes no action. The file added by Guidewire already exists in the target configuration.
GW_DELETE	delete	none	file in base not in target	If you resolve the file, the Merge Tracker takes no action. The file deleted by Guidewire no longer exists in the target configuration.
CUSTOMER_ADD	none	add	file in customer configuration only	If you resolve the file, the Merge Tracker copies the file to the target configuration if the file has not been copied there already.
CUSTOMER_EDIT	none	edit	file differs between customer and base configurations file unchanged between base and target configurations	If you resolve the file, the Merge Tracker copies the file to the target configuration if the file has not been copied there already.
CUSTOMER_DELETE	none	delete	file exists in the base and target configurations but not in the customer configuration	If you click **Delete** on the toolbar, the Merge Tracker removes the file from the target configuration. If you click **Revert to Base** on the toolbar, the Merge Tracker leaves the file in the target configuration.
BOTH_ADD	add	add	new file with matching name in both target and customer configurations (rare)	You must either merge the two versions of the file or copy your prior version of the file into the target configuration before you can resolve the file. Click
BOTH_EDIT	edit	edit	file changed in both customer and target configurations	You must either merge the two versions of the file or copy your prior version of the file into the target configuration before you can resolve the file.

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	If you resolve the file, the Merge Tracker takes no action. If you are sure you want your customized version, you can click **Copy prior customized** in the Merge Tracker toolbar to move the file to the target configuration. 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. If Guidewire has completely removed the file, review the *ClaimCenter New and Changed Guide*, release notes, and the Upgrade Diff 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. 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 filename. 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.
NO_CHANGE	none	none	file not changed from base configuration in either customer or target configurations	If you resolve the file, the Merge Tracker takes no action. The file already exists in the target configuration.

Configuration File Tree

The main panel displays the configuration file tree. Files are shown once, regardless of the number of configurations in which they exist. The number of files in each directory that match the selected change status and resolved status filters is shown in parentheses.

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.

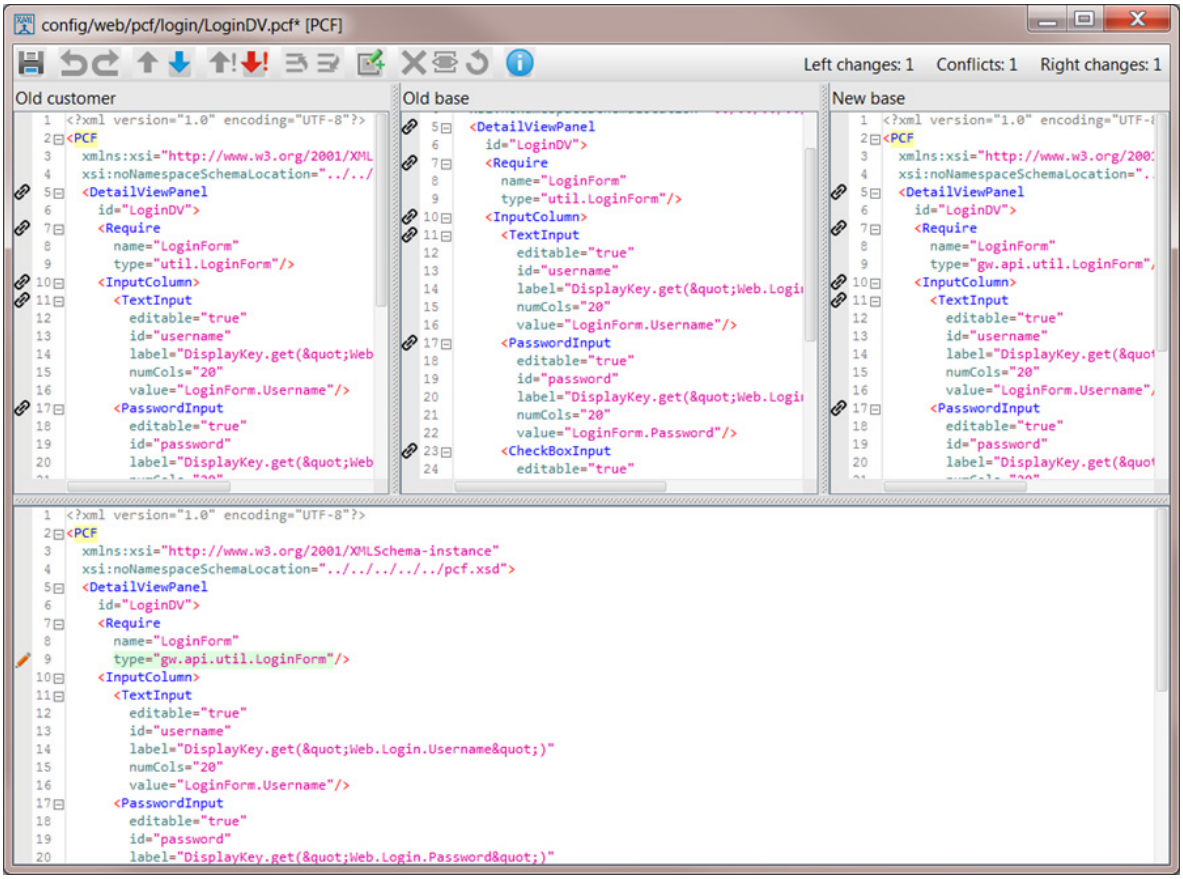
You can use the `pcf.xsd` file in the `modules` directory of the target version to validate merged PCF files.

After you are satisfied with any changes, save the file. The tool always moves files into the target configuration, unless a file is identical to the base or target version. In that case, the tool does not move the file.

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

Configuration Upgrade Smart Merge

The Smart Merge Tool is a three-way merge for Guidewire configuration files. As the name suggests, this tool is not simply an XML merge tool. Smart Merge understands the semantics of Guidewire configuration files, and uses those to assist you in the merge process.



Smart Merge Toolbar

The Smart Merge toolbar

This toolbar includes the following controls:

- **Save**
- **Undo**
- **Redo**
- **Previous Step**

- **Next Step**
- **Previous Conflict**
- **Next Conflict**
- **Move Down**
- **Move Up**
- **Add Comment**
- **Comment Out**
- **Delete Element**
- **Restore**
- **Show Help**

Smart Merge supported file types

The Smart Merge tool supports the following file types:

- Typelist definition files (`.tti`, `.tix`, `.ttx`)
- Data entity metadata files (`.eti`, `.eix`, `.etx`)
- PCF files (`.pcf`)
- Plugin descriptor files (`.gwp`)
- Display name files (`.en`)
- Workflow process files (`.#.xml`)
- Properties files (`.properties`)

Using Smart Merge in standalone mode

A standalone version of Smart Merge tool is available as the `smartmerge` script in the `upgrade/bin` directory.

The `smartmerge` script has three required arguments, *`base`*, *`left`* and *`right`*, corresponding to the three upper panes in the Smart Merge tool. The *`base`* parameter is a base configuration file. The *`left`* parameter is the customer version of the same configuration file. The *`right`* parameter is the new version of the base configuration file.

7.0.x to

Configuration Merging Guidelines

The first milestone of an upgrade project is to generate the Java and SOAP APIs (by running `gwcc regen-java-api` and `gwcc regen-soap-api`) on the target release. To generate the Java and SOAP APIs, 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 APIs or start the QuickStart application server.

You can generate the Java and SOAP APIs 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.

After you have generated the Java and SOAP APIs, 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 errors described previously, the server might fail to start due to cyclical graph reference errors. See "Identifying Data Model Issues" in the *Upgrade Guide*.

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

After 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.

Updating Data Types for Case Sensitivity

Data type definitions are case-sensitive in ClaimCenter 8.0. If you are upgrading from an early 7.0 version or a version prior to 7.0, you could have column definitions that specify a type using the wrong case. In this event, the server reports an invalid data type error during startup. If the server reports invalid data type errors, check the case of the `type` attribute for the `column` in the ETI or ETX extension file for the entity. Extension files are located in the extensions directory of the configuration module. An ETI file exists for custom entity definitions. An ETX file defines extensions to an entity provided with ClaimCenter.

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.

See the *Upgrade Diffs Report* for an inventory of differences in typekeys between the base release and the target release. To retrieve the *Upgrade Diffs Report* follow the procedure described in "Viewing Differences Between Base and Target Releases" on page 27.

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.

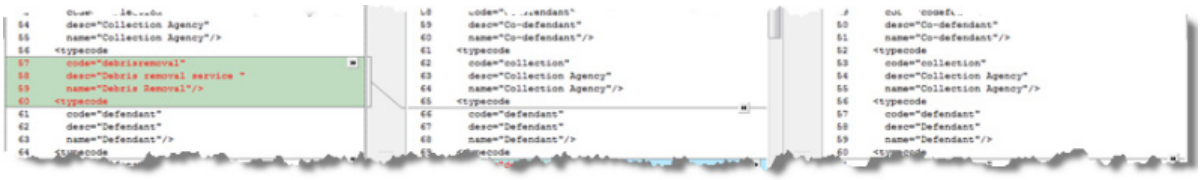
The Configuration Upgrade Tool displays most typelists you have edited in the **CUSTOMER_EDIT** filter. If your edits are simply additional typecodes, accept your version. The automated configuration upgrade may sort a typelist file. If you have also edited the typelist, then it could show the typelist in the **BOTH_EDIT** filter though the only Guidewire change was the sort.

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 9.0. 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.



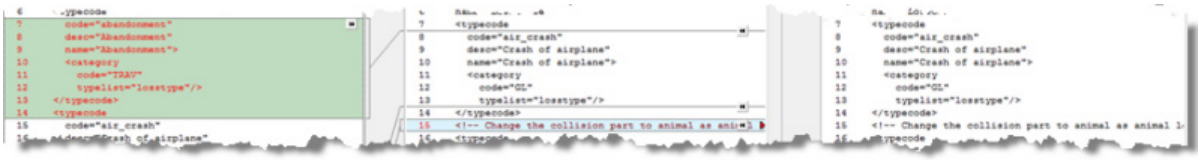
To avoid errors in enhancements, rules, and PCF files that reference the `debrisremoval` typecode, merge the typecode. If you do not want to use the new typecode, merge it into the target file and retire it by setting the `retired` attribute to `true`, as shown below.

Merging Typelists – Complex Typelists

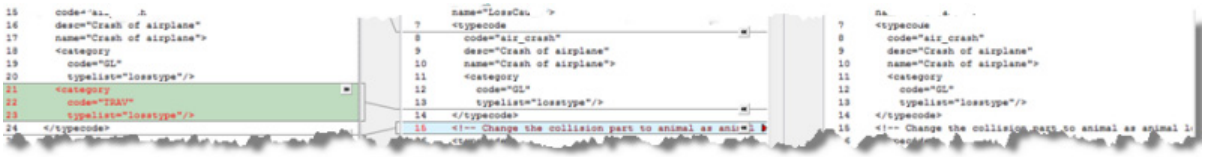
A typecode can reference typecode values from another typelist using the `<category>` subelement. If a new typecode references an existing typecode, do not merge the new typecode unless the referenced typecode is retired. Otherwise, you are defining a new relationship. If the referenced typecode is also 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 `retired` attribute of the typecode to `true`.
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 `retired` attribute of the typecode to `true`.

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.

Reviewing Shared Typekey Configuration

As of version 8.0.3, ClaimCenter enforces restrictions on the use of shared typekeys among subtypes.

Same Field Name and Typelist with Different Column

In ClaimCenter 7.0 and earlier, if a shared typekey had the same field name and typelist, and specified a different column name, ClaimCenter created only one of the typekey columns. The shared typekeys were stored in the single column. As of ClaimCenter 8.0.3, if a shared typekey with the same field name and typelist specifies a different column name, ClaimCenter creates different columns according to the specification. The database upgrade detects shared typekeys using a single column, creates the additional column, and moves the typekey data to the correct column.

Same Field and Column Names with Different Typelists

In ClaimCenter 8.0.1 and 8.0.2, a typekey on subtypes could have the same field name and column name and reference different typelists. As of ClaimCenter 8.0.3, this configuration is not allowed. The database upgrade reports an error if it detects this condition.

If you have subtypes with typekeys with the same field and column name that reference different typelists, update your data model configuration to use different column names for each typelist. The database upgrade then moves data to the new column to match the updated data model.

Adding State Typelist Extensions to Jurisdiction

ClaimCenter versions 7.0 and newer use a Jurisdiction typelist instead of a State typelist. If your environment includes custom extensions to the State typelist, move those extensions to the Jurisdiction typelist.

To move State typelist extensions to the Jurisdiction typelist

1. Open the `modules/configuration/config/extensions/State.ttx` file in your pre-upgrade starting version in a merge tool.
2. Open `modules/configuration/config/extensions/typelist/Jurisdiction.ttx` file in another panel of the merge tool.
3. Merge typecode elements from `State.ttx` to `Jurisdiction.ttx`.

Merging Lines of Business

Guidewire recommends that you leave LOB typecodes as they were in your prior version. This includes the following typelists:

- `CoverageSubtype`
- `CoverageType`
- `ExposureType`
- `LOBCode`
- `LossType`
- `PolicyType`

This also includes typelists that refer to the LOB typecodes, such as:

- `ClaimTier`
- `CostCategory`
- `CovTermPattern`
- `ExposureTier`
- `LineCategory`
- `LossPartyType`

This also includes other files that refer to the LOB typecodes, such as:

- `aggregatelimitused-config.xml`
- `ISOCoverageMap.csv`
- `policyperiod-config-xml`
- `TypeCodeMap.xml`
- `typecodemapping.xml`

If Guidewire has renamed an LOB typecode that you are using, keep the original name. References to the new name will not compile. Update these references to use the original name. Some examples of files where this may need to be done are `KeyMetrics.pcf`, `ClaimInfoBar.pcf`, and `FNOLWizard.pcf`.

Edit the `ClaimTier` and `ExposureTier` typelists to replace all instances of the new `PolicyType` typecodes with your existing `PolicyType` typecodes for the corresponding policy types. Then remove all of the `<category>` entries for policy types that you are not using. Add any new `<category>` entries as appropriate for your business needs.

Merging Entity Extensions

ClaimCenter 9.0 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.

Correcting File Naming Issues

In ClaimCenter 9.0, typelist and entity extension files must be named for the typelist or entity. In versions before 8.0, you could have an extension file name such as *Entity*_ABC.etx or *Typelist*_ABC.ttx. As of ClaimCenter 8.0, the file root name must be the entity or typelist name or the entity or typelist name followed by a dot. You can use characters after the root name to include custom name components. For example, *Entity*.ABC.etx is a valid entity extension file name. *Typelist*.ABC.ttx is a valid typelist extension file name. If you have extension files that have names that include characters other than the entity name, rename the files to put the extra characters after a dot.

Correcting Data Type References

ClaimCenter entity files must use case-sensitive references to data types. For example, setting a <column-override> to have type="shorttext" is not the same as setting type="ShortText". In this case, the former is valid while the latter is not.

Review each entity extension you have added to make sure data type references are set with the correct case.

To review and correct extension data type references

1. In Studio, expand **configuration** → **config** → **Extensions** → **Entity.**
2. Double-click each ETX file. If the file has an invalid data type reference, Studio reports that the extension field overrides validator detected a column override that refers to a non-existent data type.
3. For any such errors, select the column. Then select the correct case-sensitive **Value** for the **type** from the drop-down list.

Reviewing Optional Indexes

Guidewire often adds indexes to entities in the target configuration to improve the performance of database queries in ClaimCenter 9.0. 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 of these indexes if necessary.

Use the Configuration Upgrade Tool to resolve extension 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.

Updating setterScriptability Attributes

The setterScriptability attribute can no longer be set to external as of ClaimCenter 8.0. For any instances you have of the attribute setterScriptability set to external, change the value to all.

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 ClaimCenter directory of the target version.
2. Run the command gwb genDataDict.

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

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.

Removing Obsolete Attributes

Guidewire has removed the `deletefk` and `ondelete` attributes of the `<foreignKey>` and `<edgeForeignKey>` elements. These attributes were deprecated in an earlier major version. Now that the attributes are removed from the schema for entity definition files, if the attributes are listed, the server reports an error and does not start. Remove any occurrences of `deletefk` and `ondelete` attributes from `<foreignKey>` and `<edgeForeignKey>` elements in custom entities.

Updating Extractable Edge Foreign Keys

Guidewire has removed the `<implementsEntity>` element from `<edgeForeignKey>` and `<edgeForeignKey-override>`. In ClaimCenter 8.0, to make an edge foreign key extractable, set the Boolean `extractable` attribute on the element to `true`.

For any extractable edge foreign keys and edge foreign key overrides, delete the `<implementsEntity>` element from the key definition. Then add the attribute `extractable="true"` to the `<edgeForeignKey>` or `<edgeForeignKey-override>` element.

Handling New Currency

ClaimCenter 8.0 introduced the Japanese Yen currency. If you do not need this currency, use ClaimCenter 9.0 Studio to retire the typecode from the Currency typelist following the upgrade. If you do plan to use the Japanese Yen currency, and you have a custom exchange rate plugin, update the plugin to return a rate for the Japanese Yen.

To retire the Japanese Yen currency typecode

1. Launch Studio. From a command prompt in your ClaimCenter 9.0 installation, in the `ClaimCenter` directory, type `gwb studio`.
2. Expand **configuration** → **config** → **extensions** → **typelist**.
3. Open `Currency.ttx`.
4. Select the JPY typecode.
5. Click the checkbox for **retired** to set it to **true**.
6. Save your changes.

To update a custom exchange rate plugin

1. Launch Studio. From a command prompt in your ClaimCenter 9.0 installation, in the `ClaimCenter` directory, type `gwb studio`.
2. Expand **configuration** → **config** → **Plugins** → **registry**.
3. Open `IExchangeRateSetPlugin.gwp`.
4. Note which **Gosu Class** is the implemented exchange rate plugin.
5. Open the plugin under **gsrc**.
6. Update the `createExchangeRateSet` method to include a value for Japanese Yen in the `ExchangeRateSet` that it returns.
7. Save your changes.

Changes to the Logging API

Guidewire updated the logging API between ClaimCenter 7.0.1 and 7.0.2. Although changes to logging infrastructure were extensive, the purpose of these changes is simplification of logging usage. This document describes changes to the Guidewire logging API. If you are upgrading from a version prior to ClaimCenter 7.0.2, use this section as a guide to update your configuration files to the new logging API.

Conceptual Changes to Logging

Old API

`com.guidewire.logging.Logger`	The `Logger` class implements all logging functions. Instantiate the class with a new statement. This class is a wrapper around the Log4J `Logger` class.
`com.guidewire.logging.LoggerFactory`	The `LoggerFactory` class has two purposes. First, the class instantiates the logging infrastructure and determines the logging configuration. Second, the class is a factory that produces `Logger` instances.
`com.guidewire.logging.LoggerCategory`	The `LoggerCategory` class is a subclass of the `Logger` class. An instance of the `LoggerCategory` class behaves exactly the same way as instance of `Logger`, but `LoggerCategory` also maintains a set of static members, which are predefined loggers.
`com.guidewire.xx.system.logging.`*`XX`*`LoggerCategory` in which `xx` is a product-specific code such as `bc`, `cc`, or `pc`.	The *`XX`*`LoggerCategory` classes are application-specific subclasses of `LoggerCategory`. Normally, application-specific `LoggerCategory` classes maintain additional static `Logger` members for applications to use.

New API

`gw.pl.logging.Logger`	`Logger` was converted from a class to an interface in ClaimCenter 7.0.2. In 8.0, Logger is deprecated. You can update code to use `org.slf4j.Logger` instead of `gw.pl.logging.Logger`. The `Logger` interface provides all necessary functionality and hides implementation. This `Logger` interface explicitly prohibits certain functions that the previous `Logger` class allowed: • You cannot set logging level within the application • You cannot add nor remove appenders within the application The purpose of the `Logger` interface is to log application-specific messages. Every application component must use its own `Logger` instance to log messages relevant to the component itself. Do not perform logger management from within the component, such as defining the logging level for a logger. Instead, use the `logging.properties` file and the application interface to control logging levels and appenders.
`gw.pl.logging.LoggerFactory`	The `LoggerFactory` class retains its original functionality, but some methods have changed. This `LoggerFactory` has two purposes. First, the class instantiates the logging infrastructure and determines the logging configuration. Second, the class is a factory that produces `Logger` instances.
`gw.api.util.Logger.forCategory`	The `forCategory` method of the `Logger` class returns a Logger for the category, which is passed as a parameter to the `forCategory` method.
`com.guidewire.xx.system.logging.`*`XX`*`LoggerCategory` in which `xx` is a product-specific code such as `bc`, `cc`, or `pc`.	The *`XX`*`LoggerCategory` classes are application-specific subclasses of `LoggerCategory`. They retain their function of maintaining additional static `Logger` members for applications to use, but the static members now are instances of the `Logger` interface. You can no longer instantiate application-specific subclasses of `LoggerCategory`.
gw.api.system.*`XX`*`LoggerCategory` in which `xx` is a product-specific code such as `bc`, `cc`, or `pc`.	A mirror class to expose the logger category. It inherits all loggers defined in its `gw.pl` counterpart.

Instantiating Loggers

Old API

With the old API, you can instantiate logger instances using `Logger`, `LoggerCategory`, `LoggerFactory` or an instance of `LoggerCategory`. Any of the following statements instantiates a logger:

```
Logger logger1 = new Logger("Logger1");
Logger logger12 = new Logger(logger1, "Sublogger2");
LoggerCategory category1 = new LoggerCategory("Category1");
LoggerCategory category12 = new LoggerCategory(category1, "Subcategory2");
Logger factoryLogger1 = LoggerFactory.getInstance().getLogger("FactoryLogger1");
Logger factoryLogger12 = LoggerFactory.getInstance().getLogger(factoryLogger1,"Sublogger2");
Logger apiLogger = LoggerCategory.API;
LoggerCategory apiCategory = LoggerCategory.API;
Logger apiSubLogger = new LoggerCategory(LoggerCategory.API, "WebAPI");
LoggerCategory apiSubCategory = new LoggerCategory(LoggerCategory.API, "WebAPI");
```

New API

With the new API, you work only with instances of the `Logger` interface. You can no longer directly instantiate logger instances, so the new API supports only a few methods to obtain a logger instance:

```
// Using gw.* package
import gw.util.*;
import gw.api.system.*;
import gw.api.util.*
ILogger apiLogger = PLLoggerCategory.API;
ILogger apiSubLogger = Logger.forCategory.(PLLoggerCategory.API, "WebAPI");


// Using com.quidewire.* package
import com.guidewire.logging.*;
Logger logger1 = LoggerFactory.getLogger("Logger1");
Logger logger12 = LoggerFactory.getLogger(logger1, "Sublogger2");
Logger apiLogger = LoggerCategory.API;
Logger apiSubLogger = LoggerFactory.getLogger(PLLoggerCategory.API, "WebAPI");
```

The new API loses no functionality compared with the old API, but fewer arbitrary options exist.

Note: The `LoggerFactory` class no longer has a `getInstance()` method. The `LoggerFactory.getLogger()` method is now static.

Logging Messages

After you obtain an instance of the `Logger` with the new API, you can use the same methods as the old API to log messages. However, a new interface also allows SLF4J formatting of the messages.

Old API

```
logger.info("Started application " + appName + " with parameters " + parms.toString());
logger.info("Listening to the port " + Integer.toString(portNumber));
```

New API

```
if (wantOldStyle) {    // Old style
  logger.info("Started application " + appName + " with parameters " + parms.toString());
  logger.info("Listening to the port " + Integer.toString(portNumber));
} else {               // New style
  logger.info("Started application {} with parameters {}", appName, parms.toString());
  logger.info("Listening to the port {}", new Integer(portNumber));
}
```

Passing Loggers as Parameters

With the new API, the `LoggerCategory` class exists and has static members. However, those members are instances of the `Logger` interface instead of the `LoggerCategory` itself.

Old API

```
private LoggerCategory getApiLogger() {
  return LoggerCategory.API;
}

// ...
LoggerCategory myLogger = getApiLogger();
myLogger.debug("...");
```

New API

```
// Using gw.* package.
private gw.pl.logging.Logger getApiLogger() {
  return PLLoggerCategory.API;
}
// ...
gw.pl.logging.Logger myLogger = getApiLogger();
myLogger.debug("...");


// Using SLF4J.
private org.slf4jLogger getApiLogger() {
```

```
  return PLLoggerCategory.API;
}

// ...
org.slf4j.Logger myLogger = getApiLogger();
myLogger.debug("I am Logger {}", myLogger.toString());
```

Reviewing Changes to Multicurrency Functionality

In ClaimCenter 8 when you close an exposure or claim, ClaimCenter sets reserves on all reserve lines in all currencies to zero. This is a change in behavior from previous versions of ClaimCenter.

You might have to modify rules to allow such $0 reserve transactions. You might also want to update rules for sending transactions to downstream systems to not send transactions where the `TransactionAmount` is zero.

ClaimCenter does not enforce zeroing reserves in all currencies for exposures and claims that are already closed.

Merging Auto Death Benefit and Auto Disability Benefit

ClaimCenter 7.0.0 and 7.0.1 had Commercial Auto coverage types Auto Death Benefit (`CoverageType.BADeathCov`) and Auto Disability Benefit (`CoverageType.BADisabilityCov`). The equivalents for PolicyCenter 7.0.3 and later are one coverage type of Death and Disability with two coverage terms, Death benefit and Disability benefit. Guidewire updated ClaimCenter Commercial Auto coverages to match those of PolicyCenter. The new typekeys are `CoverageType.CADeathDisabilityCov`, `CovTermPattern.DeathBenefit`, and `CovTermPattern.DisabilityBenefit`.

If you use PolicyCenter and have Commercial Auto claims, this change might affect upgrading from ClaimCenter 7.0.0 or 7.0.1 to ClaimCenter 9.0. If your PolicyCenter and ClaimCenter configurations are not already synchronized, you can make these changes during the configuration merge. If you do make these changes, use SQL to update claims that use the Auto Death Benefit or Auto Disability Benefit typekeys to point to the new single typekey. You must also insert a `CovTerm` row to indicate if the coverage includes the Death benefit or the Disability benefit.

Adding DDL Configuration Options to database-config.xml

The configuration upgrade includes an automated step to move the database configuration from `config.xml` to `database-config.xml`. The automated step transforms most of the configuration to the 8.0 standard. However, the automated step does not transform certain DDL-related configuration settings. If you have DDL-related configuration settings for compression, Oracle SecureFile LOBs, or partitioning, recreate the configuration in the `database-config.xml` file.

DDL configuration setting changes only apply to new objects. For example, if you change an existing table from BasicFile to SecureFile LOBs, only new LOB columns will be SecureFile LOBs.

For instructions, see the following topics:

- “Configuring Compression for Databases” in the *Installation Guide*
- “Configuring ClaimCenter to Use Oracle SecureFile LOBs” in the *Installation Guide*
- “Configuring Table Partitioning for Oracle” in the *Installation Guide*

Merging Changes to Field Validators

The <ValidatorDef> element in fieldvalidators.xml accepts new attributes with ClaimCenter 8.0. All of the new attributes are optional. These attributes, the values that you can set the attributes to, and the default value of the attributes are listed in the following table:

Attribute	Values	Default	Description
validation-level	none relaxed strict	strict	The validation-level is passed to the Gosu validators. The functionality for each validation level is specific to the custom validator.
validation-type	gosu regex	regex	If validation-type is set to regex, the value of the <ValidatorDef> defines a regular expression that ClaimCenter uses to validate data entered into a field that uses the field validator. If the validation-type is set to gosu, the value of the <ValidatorDef> is a Gosu class. The Gosu class must extend FieldValidatorBase and override the validate method. See gw.api.validation.PhoneValidator for an example. Ensure that any Gosu validators that you define are low-latency for performance reasons.

Guidewire has updated some <ValidatorDef> elements in fieldvalidators.xml to use the new attributes. For <ValidatorDef> elements that you have customized, review the use of the new attribute to see if the behavior is what you want. For <ValidatorDef> elements that you have not customized, you can accept the new attributes.

Renaming PCF files According to Their Modes

In ClaimCenter 8.0, a PCF file may contain only a single mode, and must include the name of its mode, if any, in the file name. Violations of this rule produce compilation errors in Guidewire Studio. For example, if a file MyFileDV.pcf had previously defined two modes, abc and xyz, those modes must now be split into separate files, named MyFileDV.abc.pcf and MyFileDV.xyz.pcf. Even if a PCF file only contains a single mode, but that mode is not included in the file name, you must still rename the file to include the mode.

Guidewire has renamed all PCF files included in the default 8.0 configuration. However, the Configuration Upgrade Tool might not automatically fix some of your own added or changed files. In particular, take notice of **EDIT_DELETE** conflicts during the three-way merge process. Guidewire could have renamed or split apart the file based on its PCF modes rather than deleted the file. In that case, the new PCF file or files are likely to be in the same directory. Merge your changes into the new file or files.

Updating Custom Code for Moved Packages

Guidewire has moved most accessible com.guidewire packages to gw.*. See the *Upgrade Diffs* report for details on specific classes. If you have custom code that uses classes in com.guidewire packages, update the references to instead point at the implementation within the gw package.

Considering Accepting Changes to collations.xml

The `collations.xml` file includes a definition for the `GWNormalize` class. This class is used to populate generated fields to improve performance of linguistic searching. The class definition has been updated to be consistent with Java 1.7 and has other changes. If you accept these changes, it will trigger an upgrade step to update the values of every row of every table that has a column configured for linguistic searching. This is an expensive process. Only take these changes if you feel they are beneficial and can afford the increase in the upgrade duration.

Using ReserveLineInputSet in Payment and Recovery Screens

In ClaimCenter 8.0, a reserve line has its own reserving currency, distinct from the claim currency. To support this new reserve line component, Guidewire updated any pages involved in creating or editing transactions. In the case of payments and recoveries, Guidewire created a new shared input set, `ReserveLineInputSet`. PCF files involved in creating or editing payments or recoveries that included individual inputs for reserve line components (such as exposure, cost type, and cost category), include `ReserveLineInputSet` instead.

Guidewire created a new set of Gosu classes to control the behavior of `ReserveLineInputSet`. These classes replace several existing API elements. The new classes are all in the `gw.api.financials` package and have names starting with `ReserveLineInputSet`. Refer to the documentation of each class for more details.

The new `ReserveLineInputSet` contains the following inputs:

- `ReserveLine`
- `Exposure`
- `Coverage` (not editable)
- `Matter` (only shown for payments)
- `CostType`
- `CostCategory`
- `ReservingCurrency` (newly added in 8.0)

There are two major ways that this change might affect you during the upgrade:

You Have Edits in or Near the Inputs That Have Been Moved

Guidewire replaced reserve line–related inputs with a reference to `ReserveLineInputSet` in the following PCF files:

- `EditTransactionInputSet`
- `NewPaymentDetailDV`
- `QuickCheckBasicsDV`
- `RecodeRecovery`
- `RecoveryDetailDV`
- `TransferRecovery`

If you added inputs in the middle of inputs that Guidewire moved from these PCF files into `ReserveLineInputSet`, move your inputs to a different place in the original PCF file. If a custom input actually makes sense every place that the `ReserveLineInputSet` is used, you might consider adding it to `ReserveLineInputSet` itself instead. You can also follow the pattern that you see for the existing inputs and add properties to the `ReserveLineInputSetWrapper` class for any logic your new input needs.

If you have made changes to the existing inputs, carefully consider where those changes need to be made in 8.0. Purely structural or visual changes can be made directly to `ReserveLineInputSet`. Logical changes, such as validation expressions and on-change code, need to be made in one of the backing Gosu classes introduced in 8.0. The `ReserveLineInputSetWrapper` class is directly used by `ReserveLineInputSet` to provide all of its logic. `ReserveLineInputSetWrapper` itself then delegates some important parts of that logic to an instance of the `ReserveLineInputSetStrategy` abstract class. There are several concrete implementations provided, one for each situation where the `ReserveLineInputSet` is used. For example, you might need to change logic specific to creating and editing payments but not to recoveries or to recoding or transferring payments. You could either make your own subclass of `ReserveLineInputSetStrategyForPaymentCreateAndEdit` or make your changes directly to that class. The implementations for the other situations are similarly named starting with `ReserveLineInputSetStrategyFor` plus a descriptive suffix. Logic that is truly generic to all situations can go in `ReserveLineInputSetWrapper` or, rarely, `ReserveLineInputSetHelper`. See the documentation for those classes for guidance. All of the backing classes for `ReserveLineInputSet` are in the `gw.api.financials` package.

`NewPaymentDetailDV.pcf` has been renamed to `NewPaymentDetailDV.default.pcf`, so your changes might not be merged automatically. See "Renaming PCF files According to Their Modes" on page 83. However, Guidewire deleted its other modes, `SingleLineItemAmount` and `ReflectedSingleLineItemAmount`, because they were not being used at all. So the default mode is the only one remaining.

You Have Used the Removed APIs in Other Code

Guidewire removed several existing Java APIs and replaced them with Gosu classes. There is not a direct one-to-one correspondence between the old APIs and the new classes. If you are using the old APIs outside of the PCF files they were intended for, study the documentation of the new classes to understand how to use them. The classes have names starting with `ReserveLineInputSet` and are in the `gw.api.financials` package.

Notice that there is a careful separation of responsibilities among the classes. The various subclasses of `ReserveLineInputSetStrategy` contain most of the interesting logic for filtering reserve lines and exposures. In particular, the `ReserveLineInputSetStrategyForPaymentBase` subclass is responsible for evaluating which claims, exposures, and reserve lines can have new payments created on them. The other classes, `ReserveLineInputSetHelper` and `ReserveLineInputSetWrapper`, are more tailored to the user interface and are therefore less likely to be useful in other situations.

The removed APIs in question are:

- `gw.api.financials.NewRecoverySetHelper`
- `gw.api.financials.EditTransactionHelper`
- `com.guidewire.cc.web.financials.EditPaymentHelperUtil`
- `gw.api.financials.CheckWizardInfo:canCreateNewClaimLevelReserve(entity.Payment)`
- `gw.api.financials.CheckWizardInfo:getPayableExposures(entity.Payment)`
- `gw.api.financials.CheckWizardInfo:getPayableReserveLines(entity.Payment)`
- `gw.api.financials.CheckWizardInfo:getPossibleCostCategories(entity.Payment)`
- `gw.api.financials.CheckWizardInfo:getPossibleCostTypes(entity.Payment)`
- `gw.api.financials.CheckWizardInfo:getReserveLinePickerOptionLabel(entity.ReserveLine)`
- `gw.api.financials.CheckWizardInfoBase:HelperUtil`
- `gw.api.financials.CheckWizardInfoBase:setHelperUtil(com.guidewire.cc.web.financials.EditPaymentHelperUtil)`

Before investigating usages of these APIs that you find in your code, make sure that all files have been merged successfully. In particular, `NewPaymentDetailDV.pcf` has been renamed to `NewPaymentDetailDV.default.pcf`, so your changes might not be merged automatically. See "Renaming PCF files According to Their Modes" on page 83. However, its other modes, `SingleLineItemAmount` and `ReflectedSingleLineItemAmount`, have been deleted because they were not being used at all. The default mode is the only one remaining.

Reviewing Replacement of Fields and Roles with Service Requests

There are several places in the FNOL screens and the incident detail screen where ClaimCenter previously set a field or a role and now creates `ServiceRequest` entities instead. Review these changes and confirm that you do not need the field or role set as it was in earlier versions. In these cases, the fields and roles have not been removed, but all usages have been removed from the default 8.0 configuration. These are the roles and fields that still exist but are no longer used:

Fields:

- `DwellingIncident.DebrisRemovalInd`
- `DwellingIncident.EMSInd`
- `VehicleIncident.LocationInd`
- `PropertyIncident.AppraisalFirstAppointment`
- `PropertyIncident.InspectionRequired`
- `PropertyIncident.WhenToView`
- `VehicleIncident.VehTowedInd`
- `VehicleIncident.RentalAgency`
- `VehicleIncident.RentalRequired`
- `VehicleIncident.RepWhereDisInd`

The fields `RentalRequired` and `RentalAgency` are no longer used in the 8.0 base configuration. However, in 7.0 and earlier, they were used in conjunction with fields that are still used in 8.0: `RentalBeginDate`, `RentalEndDate`, `RentalDailyRate`, and `RentalReserveNo`. Because of this, upgraded claims may have data in these non-deprecated fields that is not visible in the user interface. The presence of a service, rather than the value of `RentalRequired`, determines whether these fields are shown.

Roles:

- `ems`
- `debrisremoval`
- `fnolassessor`
- `repairshop`
- `TowingAgcy`

If your configuration relies on any of these fields or roles, you have several options:

- Keep the base configuration changes and manually create services on an as-needed basis for upgraded claims.
- Update the PCF files to reinstate the user interface that exposed these fields. Note that this user interface will be somewhat redundant with the new user interface that models the same information using services.
- Write upgrade triggers to automatically create services for data where these fields and roles were used.

  Note: The `DateCalculator` interface in the `gw.api.metric` package has changed. Guidewire renamed the methods `daysBetween` and `addDays` to `timeBetween` and `addTime` to accommodate some service request metrics that are calculated in hours instead of days. The functionality has not changed.

Reviewing Change to Aggregate Limits Screen

The **Limit Type** field was removed from the Aggregate Limits screen. The limit type remains the same for all limits defined on a policy, so ClaimCenter does not need to capture this on a per-limit basis.

You can add the **Limit Type** field back to `AggregateLimitDetailDV.pcf` with possible values of reported date or loss date, as defined in the `LimitType` typelist. ClaimCenter will honor those values correctly if you add the **Limit Type** field.

Merging Display Properties

The Configuration Upgrade Tool updates display properties files, such as `display.properties`, as described in "Upgrading Display Keys" on page 63 to create a merged file with the extension `.merged`. You could have conflicts in the files if you have a different number of parameters for a key than the 8.0 version or if you have a different value.

If the number of parameters differs from the 8.0 version, match your parameter set to the 8.0 version of the key.

If the value is different, choose which value you want to use in your ClaimCenter configuration.

Merge changes into `display.properties.merged`. When you save the file, the Configuration Upgrade Tool saves it to the configuration module without the `.merged` extension.

If you have added locales, you can export a full list of display keys and typelists from the default ClaimCenter 9.0 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 94.

In ClaimCenter 8.0, Studio trims trailing spaces from display keys by default. You can modify this behavior using the following procedure:

1. Click **File** → **Settings**.
2. Under **IDE Settings** click **Editor**.
3. Under **Other**, change the value of **Strip trailing spaces on Save** to **None**.
4. Click **OK**.

Merging Other Files

In some cases, you cannot effectively merge the differences between files 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.

During startup, ClaimCenter 9.0 reports a warning message if you have configuration parameters defined in `config.xml` that ClaimCenter 9.0 does not use. ClaimCenter ignores any unused parameters. You might have old parameters in `config.xml` that ClaimCenter does not use. If ClaimCenter 9.0 reports that there are unknown parameters specified, remove these parameters from `config.xml`.

If your installation contains a language that is not one of the core Guidewire-supported languages in the base configuration, in `config.xml` copy the value of `DefaultApplicationLocale` to `DefaultApplicationLanguage`. The core Guidewire-supported languages in the base configuration are U.S. English, Italian, German, Spanish, French, Chinese, and Japanese.

Fixing Gosu Issues

Review additions and changes to Gosu code in the ClaimCenter New and Changed Guide. Update your Gosu code for these changes. Use the procedures in this topic to detect and fix these issues.

Gosu Case Sensitivity

ClaimCenter 8.0 has strict case-sensitivity for Gosu code.

To detect and fix case-mismatch issues

1. Right-click a folder in the **Project** pane and select **Analyze** → **Run Inspection by Name...**.

   Note: Do not select the whole project as the inspection is resource intensive.
2. Enter `case` and double-click **Name is referenced with improper case**.
3. In the dialog, set **Inspection scope** to **Directory**.
4. Deselect **Include test sources**.
5. Click **OK**.
6. In the **Results** pane, expand **Case mismatch issues**, if present.
7. Right-click the **Name is referenced with improper case** issue type, and click **Apply Fix 'Case mismatch issues'**.
8. Click the **Save All** icon.
9. Repeat this procedure for the selected folder until no case mismatch issues are reported or the count stops dropping. It might not drop all the way to zero. Keep a record of any folders that do not reach zero errors.
10. Continue this process for all folders containing files with Gosu code.
11. If any folders have an error count above zero, and the count is not dropping after you apply the fix, compile the project to detect other errors.

Inequality Operator

The inequality operator <> is no longer valid and must be replaced with `!=`.

To detect and fix the obsolete inequality operator

1. Right-click a folder in the **Project** pane and select **Analyze** → **Run Inspection by Name...**.

   Note: Do not select the whole project as the inspection is resource intensive.
2. Enter `The <>` and double-click **The <> operator is obsolete**.
3. In the dialog, set **Inspection scope** set to **Directory**.
4. Click **OK**.
5. In the **Results** pane, expand **Equality issues**, if present.
6. Right-click issue type **The <> operator is obsolete**, and click **Apply Fix 'Equality issues'**.
7. Click the **Save All** icon.
8. Repeat this procedure for the selected folder until no equality issues are reported or the count stops dropping. It might not drop all the way to zero. Keep a record of any folders that do not reach zero errors.
9. Continue this process for all folders containing files with Gosu code.

10. If any folders have an error count above zero, and the count is not dropping after you apply the fix, compile the project to detect other errors.

Ambiguous Method Calls

Previous versions of Gosu reported a warning on ambiguous method calls. Ambiguous method calls can hide a logical bug in your code. Previously, the Gosu compiler selected the best matching method to remove ambiguity. For ClaimCenter 8.0, ambiguous calls are now an error instead of a warning. Studio now has a code inspection to identify and optionally fix any ambiguous code to previous Studio behavior. This inspection is disabled by default. To find and fix potential logical errors, Guidewire recommends that you run the inspection and carefully individually analyze every ambiguous call before applying any proposed fix.

To detect and fix ambiguous method calls

1. Right-click a folder in the **Project** pane and select **Analyze** → **Run Inspection by Name...**.

   Note: Do not select the whole project as the inspection is resource intensive.

2. Enter `The method` and double-click **The method call is ambiguous, it can be fixed by adding casts.**
3. In the dialog, set **Inspection scope** to **Directory.**
4. Deselect **Include test sources.**
5. Click **OK**.
6. In the **Results** pane, expand **The method call is ambiguous, it can be fixed by adding casts**, if present.
7. Analyze and fix any ambiguous method calls that are reported.
8. Repeat this procedure for the selected folder until no ambiguous method call issues are reported or the count stops dropping. It might not drop all the way to zero. Keep a record of any folders that do not reach zero errors.
9. Continue this process for all folders containing files with Gosu code.
10. If any folders have an error count above zero, and the count is not dropping after you apply the fix, compile the project to detect other errors.

Nested Comments

Gosu supports nested comments. The purpose of nested comments is to quickly comment out large swaths of code temporarily while avoiding compiler errors whenever the enclosed code contains comments.

In earlier releases, the Gosu compiler searched only for "`*/`" after encountering a comment that opened with "`/*`". This behavior permitted developers to include dividing lines within lengthy comments, like the following example.

```
//******************************************************
```

In ClaimCenter 9.0, the Gosu compiler searches for "`/*`" after encountering a comment that opens with "`/*`" in case the comment body contains a nested comment. Because the comment line in the preceding example begins with "`/*`", the compiler begins searching for the close of the nested comment and never finds one.

Following an upgrade to ClaimCenter 9.0, the Gosu compiler may produce the following error message:

```
unclosed comment
This occurs in multiple-line comments that use the open and close comment marks "/*" and "*/" if the
comment body contains the character sequence "/*.
```

To resolve unclosed comment errors

1. If the Gosu compiler reports the unclosed comment error, open the source file in Studio.

2. Rewrite any comments that inadvertently include the character sequence "`/*`" within the body of comments. In the preceding example, you could avoid the problem by inserting a space between the slash and the asterisk or by changing to a sequence of characters other than asterisks.

   If there are a number of errors for one source file, consider opening the source in the pre-upgrade version of Studio. Then you can compare the commented sections between the old and new Gosu behavior.

3. Compile the project to find any further errors.

Upgrading Rules to ClaimCenter 9.0

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/configuration/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 to a ClaimCenter 9.0 Rules folder within `modules/configuration/config/rules`. Use Studio to update your rules. You can use the rules in the ClaimCenter 9.0 folder as a comparison. Compare your custom rules to the new default 9.0 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 9.0 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 9.0 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 `ClaimCenter` directory of your starting version.

4. Enter the following command:

   ```
   gwb genRuleReport
   ```

   This command creates a rule repository report XML file in `build/rules`.

5. Append the starting version number to the XML file name.

6. Restore moved directories to the starting version.

7. Install files for a fresh ClaimCenter 9.0 version. This is a separate configuration from the target configuration that you have merged. This version will only contain the default rules provided with ClaimCenter 9.0.

8. Navigate to the `ClaimCenter` directory of the new ClaimCenter 9.0 version.

9. Enter the following command:

   ```
   ClaimCenter
   ```

   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. 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.

In Studio, you can compare custom rules to default ClaimCenter 9.0 rules by opening the default rules in the ClaimCenter 9.0 directory within **configuration** → **config** → **Rule Sets**. When you have finished updating all of your custom rules, delete the ClaimCenter 9.0 rules directory from `modules/configuration/config/rules`.

The ClaimCenter 9.0 default rules are enabled because some features depend on these rules. See “Rules Required for Key 8.0 Features” on page 91.

Claim PreUpdate Reinsurance Rules

The claim pre-update reinsurance rules introduced in ClaimCenter 6.0 have been superseded by new rules in ClaimCenter 7.0.1 and higher.

If you want to continue to use the default earlier versions of the reinsurance rules, copy `ClaimCenter/tmp/cfg-upgrade/modules/cc/config/rules/Preupdate/ClaimPreupdate_dir/CPU17000Reinsurance_dir` to `ClaimCenter/modules/configuration/config/rules/Preupdate/ClaimPreupdate_dir/CPU17000Reinsurance_dir` in the target configuration.

If you have custom versions, copy `ClaimCenter/tmp/cfg-upgrade/modules/configuration/config/rules/Preupdate/ClaimPreupdate_dir/CPU17000Reinsurance_dir` to `ClaimCenter/modules/configuration/config/rules/Preupdate/ClaimPreupdate_dir/CPU17000Reinsurance_dir`. This copies any custom ClaimCenter 6.0 reinsurance rules into ClaimCenter 9.0. Use Studio to disable the ClaimCenter 9.0 reinsurance rules and enable the ClaimCenter 6.0.x reinsurance rules. Review the default claim pre-update reinsurance rules in Studio to determine if you want to include any of the changes to the default rules to your custom versions.

Rules Required for Key 8.0 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. To find a rule, right-click **Rule Sets** in Studio and click **Find in Path...**. Then enter the rule identifier, such as CCL02000, and click **Find**.

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. To find a rule, right-click **Rule Sets** in Studio and click **Find in Path...**. Then enter the rule identifier, such as CCL02000, and click **Find**.

Weighted Workload Assignment

Assignment\DefaultGroupClaimAssignmentRules_dir

DGC00500Balancedworkloadwithingroup.gr

Assignment\DefaultGroupExposureAssignmentRules_dir

DGE00500Balanceworkloadwithingroup.gr

Preupdate\ClaimPreupdate_dir

CPU30000WorkloadAssignmentBalancing.gr

CPU30000WorkloadAssignmentBalancing_dir\CPU30100ClaimClosed.gr

CPU30000WorkloadAssignmentBalancing_dir\CPU30200ClaimReassignment.gr

CPU30000WorkloadAssignmentBalancing_dir\CPU30300ClaimReopened.gr

CPU30000WorkloadAssignmentBalancing_dir\CPU30400NewClaim.gr

CPU30000WorkloadAssignmentBalancing_dir\CPU30500ClaimWorkloadAffected.gr

Preupdate\ExposurePreupdate_dir

EPU10000WorkloadAssignmentBalancing.gr

EPU10000WorkloadAssignmentBalancing_dir\EPU10100ExposureClosed.gr

EPU10000WorkloadAssignmentBalancing_dir\EPU10200ExposureReassignment.gr

EPU10000WorkloadAssignmentBalancing_dir\EPU10300ExposureReopened.gr

EPU10000WorkloadAssignmentBalancing_dir\EPU10400NewExposure.gr

EPU10000WorkloadAssignmentBalancing_dir\EPU10500ExposureWorkloadAffected.gr

Preupdate\UserPreupdate_dir

UPU00005UserWorkloadResyncOnStatusChange.gr

Service Requests

Assignment\DefaultGroupServiceRequestAssignmentRules_dir

DGS01000Default.gr

Assignment\GlobalServiceRequestAssignmentRules_dir

GSA01000Default.gr

Validation\ClaimClosedValidationRules_dir

CCV08000OpenServiceRequests.gr

Validation\ServiceRequestValidationRules_dir

SRVR01000ServiceRequestHistoryAccurate.gr

SRVR02000ServiceRequestInvoiceCurrencies.gr

Validation\TransactionSetValidationRules_dir

TXV17000ReserveLinescompatiblewithServiceRequests.gr

TXV18000CheckcurrenciescompatiblewithServiceRequestInvoices.gr

Contact System Integration

The rules for Contact System Integration have changed in 8.0 with the new `ContactSystemPlugin` interface. The old rules are still present but disabled, unless there has been no significant change due to the new API.

The disabled rules are:

- EFR08110 - Update or Add
- EFR08115 - Tags Updated

The new rules are:

- EFR08160 - Update
- EFR08200 - Not Linked
- EFR08210 - Update or Add

Vendor Portal

EventMessage\EventFired_dir

EFR10000VendorNotifications.gr

Rules Required for Key Features Introduced in Releases Before 8.0

Contact Automatic Synchronization

CCL02000 – Suspend AutoSync for Related Contacts

COP03000 – Add default tags

COP01000 – Update Check Address

Catastrophe Bulk Association

CPU09000 – Related to Catastrophe

CPU13000 – Catastrophe History

CLV10000 – Catastrophe

CPU19000 – Geocode Catastrophe Claims

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

CPU17000 – RI Notifications

ISO Validation

CLV09000 – ISO Validation

Translating New Display Properties and Typecodes

ClaimCenter 9.0 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. Export display keys by running the following command from your ClaimCenter 9.0 environment `ClaimCenter` directory:
   ```
   gwb exportL10ns -Dexport.file="translation_file" -Dexport.locale="language to export"
   ```
2. Open the exported translation 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.
3. Specify localized values for the untranslated properties.
4. Save the updated file.
5. Import the updated file by running the following command from your ClaimCenter 9.0 environment `ClaimCenter` directory:
   ```
   gwb importL10ns -Dimport.file="translation_file" -Dimport.locale="language to import"
   ```
   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 ContactManager), 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 `gwb studio` command in the `ClaimCenter` directory.
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.

ClusteringComponent Class Removed

The `ClusteringComponent` class has been removed.

Replace usages such as:

```
ServerDependencies.getClusteringComponent().isClusteringEnabled()
```

With:

```
ClusterEnabledInfo.isClusteringEnabled().
```

Validating the ClaimCenter 9.0 Configuration

This topic includes procedures to validate the upgraded configuration.

Using Studio to Verify Files

You can use Studio to verify classes and enhancements, including libraries, PCF files, rules, and typelists without having to start ClaimCenter. Do not start ClaimCenter at this point. Studio can run without connecting to the application server.

To validate Studio resources

1. Start Guidewire Studio by running `gwb studio` from the `ClaimCenter` directory.
2. Click **Analyze** → **Inspect Code...**.
3. Set the **Inspection scope** to **module 'configuration'**.
4. Click OK. Studio runs inspections to identify incorrect Gosu syntax, issuing either a warning or an error.
5. Correct all identified errors with Studio. You can defer fixing warnings.

Starting ClaimCenter and Resolving Errors

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" in the *Installation Guide* and "Deploying ClaimCenter to the Application Server" 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, such as 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 9.0

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” 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, then upgrade, build, and deploy each individually before attempting to reintegrate 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` 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. Building and deploying a WAR or EAR file copies the JAR file into the appropriate place for it to be accessed by ClaimCenter. JAR files in this location survive the upgrade process.

chapter 6

Upgrading the ClaimCenter 7.0 Database

To upgrade your ClaimCenter Database, refer to “Upgrading the ClaimCenter 7.0 Database” in the *ClaimCenter Upgrade Guide*, which is included with the main ClaimCenter documentation set.