Guidewire InsuranceSuite Configuration Upgrade Tools

Guide

Release 3.1.0

GUIDEWIRE
Adapt and succeed™

Product Name: Guidewire InsuranceSuite Configuration Upgrade Tools
Product Release: 3.1.0
Document Name: Guide
Document Revision: 23-July-2020

Contents

Guidewire Documentation

Conventions in this document

Text style	Meaning	Examples
italic	Indicates a term that is being defined, added emphasis, and book titles. In monospace text, italics indicate a variable to be replaced.	A *destination* sends messages to an external system. Navigate to the InsuranceSuite Configuration Upgrade Tools installation directory by running the following command: `cd` *`installDir`*

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

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

Support

For assistance, visit the Guidewire Community.

Guidewire customers

`https://community.guidewire.com`

Guidewire partners

`https://partner.guidewire.com`

chapter 1

Release notes

This topic describes changes in InsuranceSuite Configuration Upgrade Tools 3.1.0.

Improvements and resolved issues

Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
UT-715	Fixed the following issues with automatic merging of changes in the `display.properties` file: • Unmodified display keys did not stay in their original position • Spaces around the equals sign (=) were removed
UT-723	Restored a preupgrade step, **ExternalEntityCoercion**, which replaces implicit coercion for external entities with explicit conversion calls. The step was removed in InsuranceSuite Configuration Upgrade Tools 1.25.0.

chapter 2

Overview of the InsuranceSuite Configuration Upgrade Tools

The InsuranceSuite Configuration Upgrade Tools are a set of utilities that help you with configuration upgrade.

How to get the InsuranceSuite Configuration Upgrade Tools

In InsuranceSuite release 9 and later, the tools are not included in InsuranceSuite applications. You can get them from the Guidewire Community..

In InsuranceSuite release 8 and earlier, the tools are included in InsuranceSuite applications. These legacy tools are only used in a limited scope when you upgrade configuration from release 7 or earlier.

What is included in the InsuranceSuite Configuration Upgrade Tools

The InsuranceSuite Configuration Upgrade Tools include the following tools:

Preupgrade

The preupgrade tool prepares a 7.x or 8.x configuration for the automated upgrade process by performing a number of code cleanup and formatting steps. Changes made by the preupgrade tool are destructive and cannot be rolled back because the tool modifies your original customized configuration. For example, if you are upgrading an InsuranceSuite application from release 8 to 10, the preupgrade tool will modify the configuration of the InsuranceSuite application 8.0.

Upgrade

The upgrade tool prepares your configuration for the manual merge process by performing a number of automated steps. Changes made by the upgrade tool are safe because it copies your original customized configuration to a temporary location and runs a series of automated upgrade steps on this temporary folder.

Merge Tracker

The Merge Tracker tool is a Guidewire Studio plugin that tracks progress and assists you in completing the manual part of the configuration upgrade process. Merge Tracker gives you a central location to view and work with the configuration files:

- Shows a complete list of all configuration files, along with the status of each file.
- Provides filters for viewing a portion of the configuration.
- Provides a central location to launch tools, such as Smart Merge, Smart Diff, and upgrade inspections.

Smart Merge

The Smart Merge tool is a three-way merge tool that understands the semantics of Guidewire configuration files, and uses that understanding to assist you in the merge process.

Smart Diff

The Smart Diff tool is a comparison tool that understands the semantics of Guidewire configuration files.

Upgrade inspections

Upgrade inspections are a set of Guidewire Studio inspections that identify issues after you complete merging and resolving changes. The inspections search the new customized configuration in the target environment for constructs that need to be upgraded. In Merge Tracker, you can run the upgrade inspections and then fix the issues manually or you can run the upgrade inspections with automatic fixes. From the command prompt, you can run code and upgrade inspections and save the results to a CSV file. You cannot apply fixes from the command prompt.

chapter 3

How the InsuranceSuite Configuration Upgrade Tools work

Environments and configurations involved in the configuration upgrade process

When you upgrade the configuration, you work with two environments:

Source environment

The installation directory of your InsuranceSuite application.

Target environment

The installation directory of the new release of the InsuranceSuite application.

For example, when you upgrade from release 8 to release 10, the source environment is the installation directory for release 8 and the target environment is the installation directory for release 10.

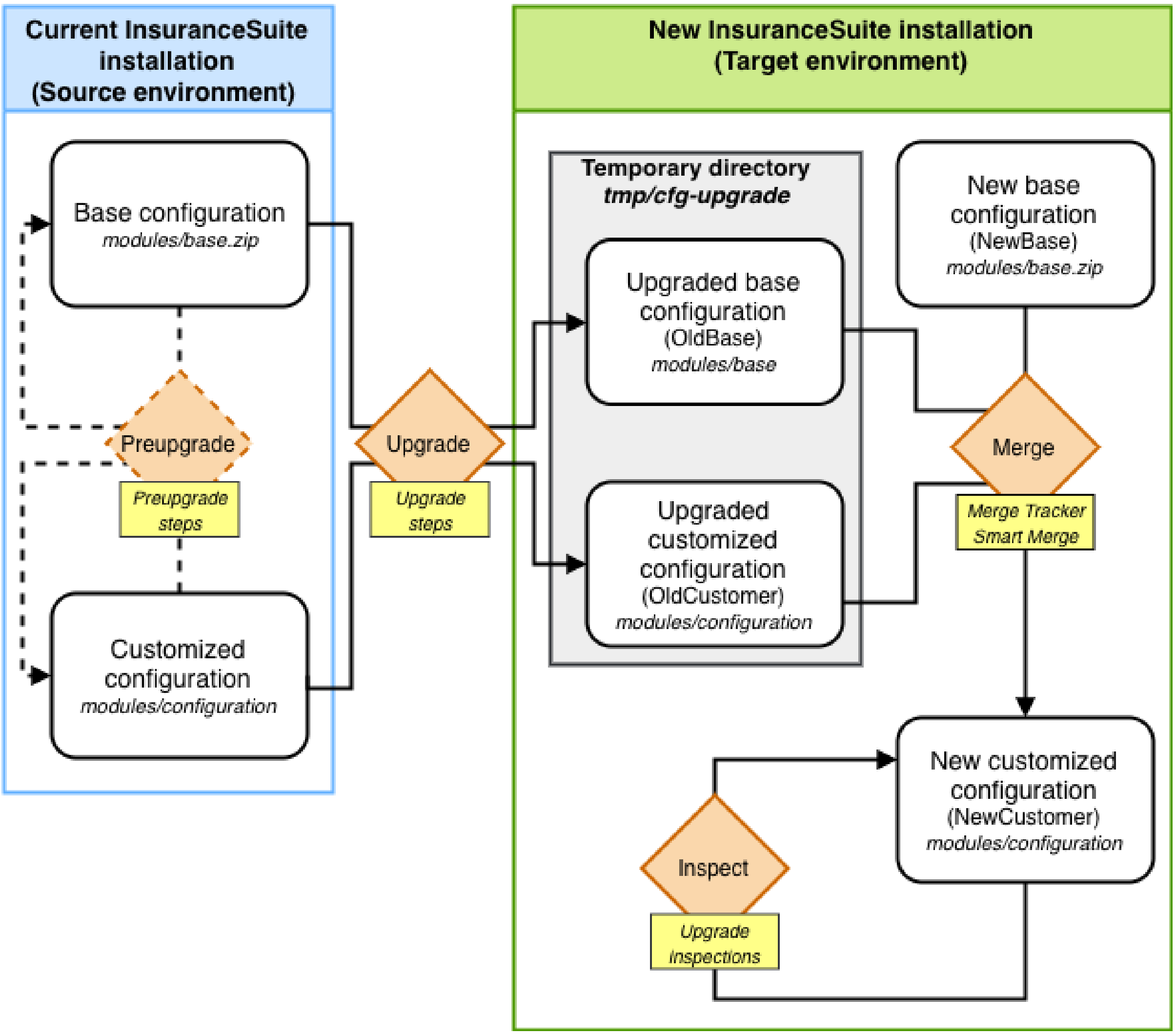
The configuration upgrade process involves the following configurations.

Name	Name in the InsuranceSuite Configuration Upgrade Tools	Description	Location
Base configuration	None	A copy of the original unedited configuration delivered by Guidewire.	`modules/base.zip` in the source environment
Customized configuration	None	The configuration with your custom changes that you are currently using.	`modules/configuration` in the source environment
Upgraded base configuration	OldBase	A temporary copy of the base configuration that contains changes made by the InsuranceSuite Configuration Upgrade Tools.	`tmp/cfg-upgrade/modules/base` in the target environment
Upgraded customized configuration	OldCustomer	A temporary copy of the customized configuration that contains changes made by the InsuranceSuite Configuration Upgrade Tools.	`tmp/cfg-upgrade/modules/configuration` in the target environment

Name	Name in the InsuranceSuite Configuration Upgrade Tools	Description	Location
New base configuration	NewBase	The new version of the unedited configuration delivered by Guidewire.	`modules/base.zip` in the target environment
New customized configuration	NewCustomer	The configuration that you get after the upgrade process is complete. It contains changes from your customized configuration and improvements delivered by Guidewire in the new base configuration.	`modules/configuration` in the target environment

Upgrade from release 8 or later

The following diagram shows how the InsuranceSuite Configuration Upgrade Tools handle a configuration upgrade of an InsuranceSuite application 8 or later.



The process shown in the diagram is as follows:

1. For upgrades from releases prior to 9, you run preupgrade steps that prepare your configuration for the upgrade process. These changes are destructive and cannot be rolled back because the tool modifies your original customized configuration.
2. You run upgrade steps that:
   a. Create a `tmp/cfg-upgrade` directory in the installation directory of the new release of the InsuranceSuite application.
   b. Copy the base configuration and the customized configuration to the temporary directory.
   c. Perform a number of transformations to upgrade the copied base and customized configurations in the temporary directory to the new release of the InsuranceSuite application.
3. You use Merge Tracker and Smart Merge to merge and resolve changes from the upgraded customized configuration with changes from the new base configuration. The upgraded base configuration is used as baseline for identifying changes and conflicts. The result of the merge and resolve process is the new customized configuration.
4. You run upgrade inspections on the merged configuration and apply fixes to get the configuration that is production-ready.

chapter 4

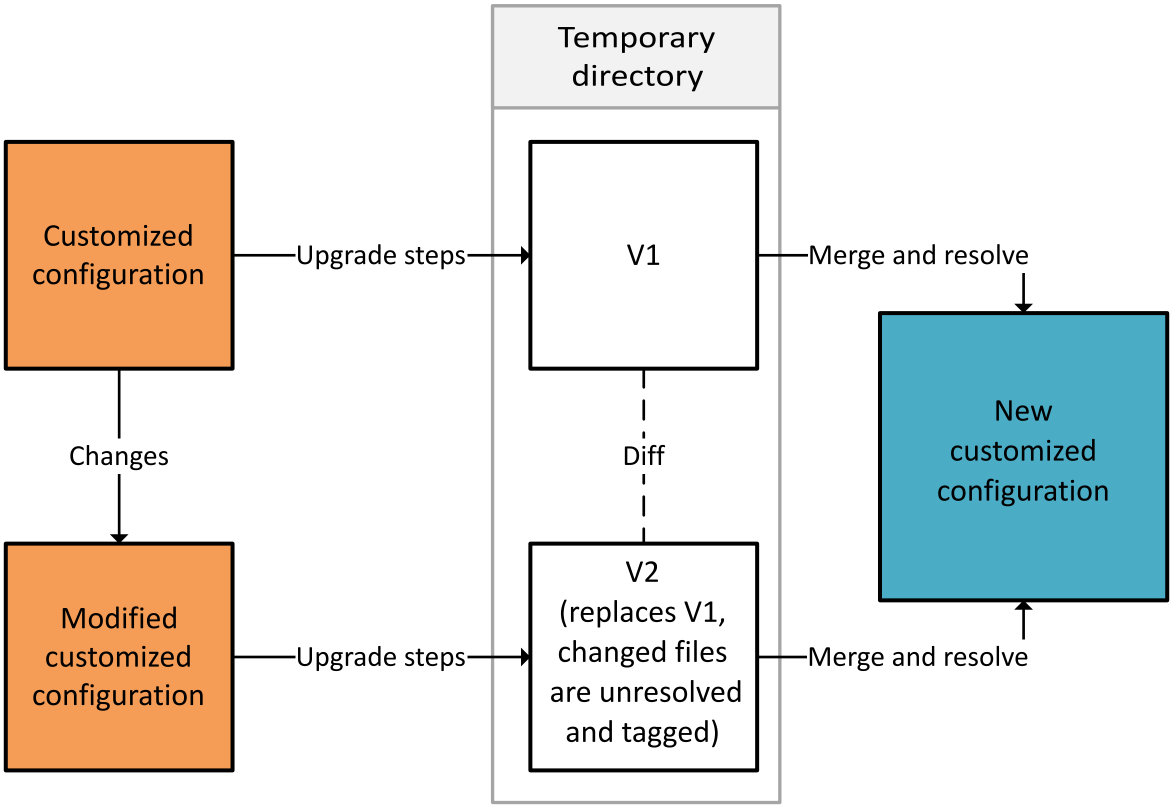
Handling changes to the configuration during an upgrade

This topic describes how to handle changes to the customized configuration during the configuration upgrade process.

Process overview

1. You run upgrade steps. A temporary directory is created.
2. You launch Merge Tracker and then merge and resolve files. In the meantime, the customized configuration changes.
3. You run the upgrade steps again. The existing temporary directory is deleted and then a new one is created.
4. You go back to merging and resolving files in Merge Tracker.

You may need to repeat steps 3 and 4 because the customized configuration can change multiple times during the upgrade process.


Temporary directory
Customized configuration
Upgrade steps
V1
Merge and resolve
New customized configuration
Changes
Diff
Modified customized configuration
Upgrade steps
V2 (replaces V1, changed files are unresolved and tagged)
Merge and resolve

Tools

The InsuranceSuite Configuration Upgrade Tools allow you to run upgrade steps again without losing your merging progress. Each configuration file in the upgraded customized configuration has a status file that contains a CRC32 checksum of the file content. The status files are located outside the temporary directory. When you launch Merge Tracker, it compares the checksums in the status files with current checksums. If the **Unresolve file when changes in upgraded customized configuration are detected** is enabled in Merge Tracker settings, all the resolved files that changed are unresolved automatically and tagged as **changed-after-resolution** in Merge Tracker. For more information on status files, see “Merge progress tracking” on page 72.

Optionally, you can add the temporary directory to a version control system, such as Git, to have more control over changes made to the customized configuration. When files in the temporary directory are tracked, you can identify changes that happened between subsequent runs of upgrade steps.

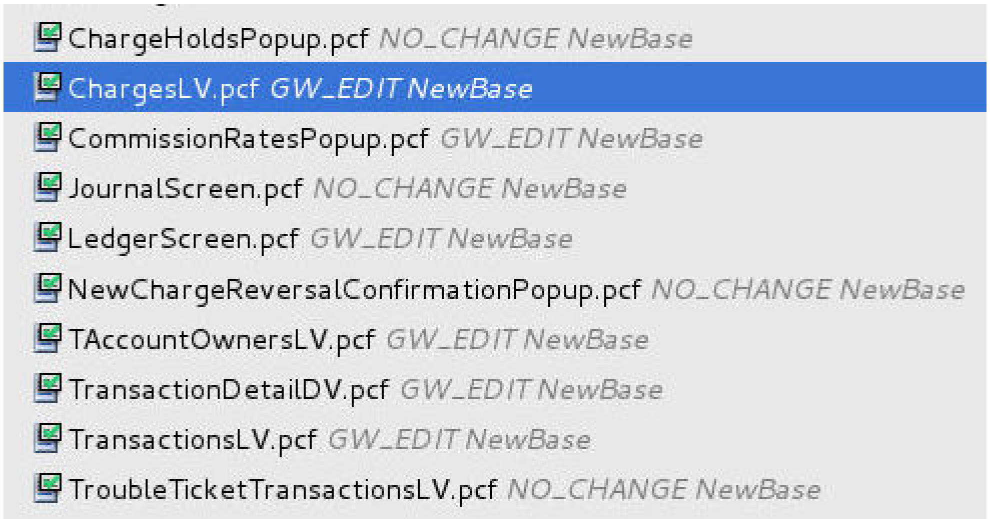
Example

The process described in this section is an example of handling changes to the customized configuration that are made during the upgrade. The procedure assumes that the InsuranceSuite Configuration Upgrade Tools are installed in the target environment and configured, the **Unresolve file when changes in upgraded customized configuration are detected** is enabled in Merge Tracker settings, and Git is used as the source control system.

1. Run the `upgrade` command. The temporary directory is created in the target environment.
2. Go to the target environment and add the `tmp` directory to Git.

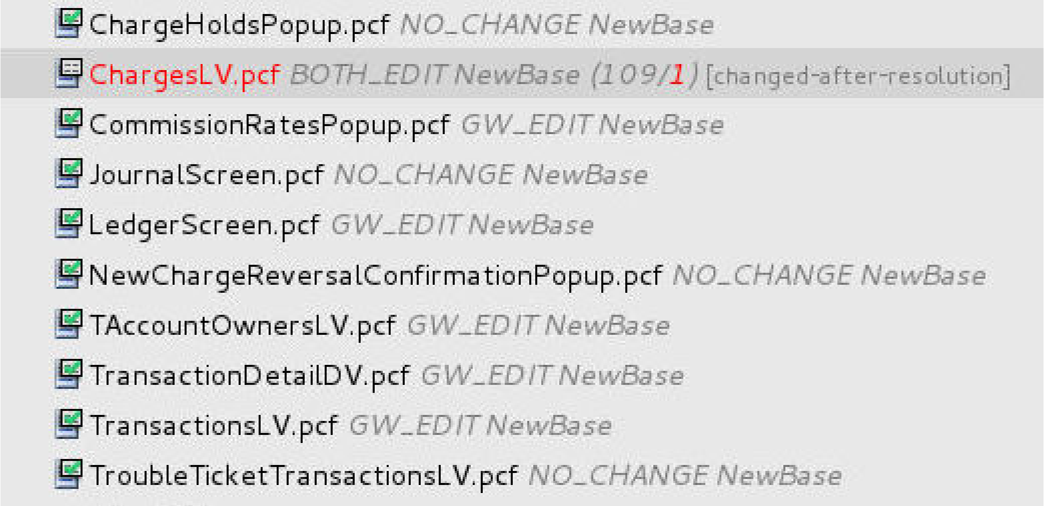
```
git init
git add tmp
git commit -m 'Upgraded configuration initial version'
```

3. Run Guidewire Studio in the target environment. When it starts, open Merge Tracker.
4. Merge and resolve files as needed. The files are marked as **Resolved** in Merge Tracker.



ChargeHoldsPopup.pcf *NO_CHANGE NewBase*
ChargesLV.pcf *GW_EDIT NewBase*
CommissionRatesPopup.pcf *GW_EDIT NewBase*
JournalScreen.pcf *NO_CHANGE NewBase*
LedgerScreen.pcf *GW_EDIT NewBase*
NewChargeReversalConfirmationPopup.pcf *NO_CHANGE NewBase*
TAccountOwnersLV.pcf *GW_EDIT NewBase*
TransactionDetailDV.pcf *GW_EDIT NewBase*
TransactionsLV.pcf *GW_EDIT NewBase*
TroubleTicketTransactionsLV.pcf *NO_CHANGE NewBase*

5. After changes are made in the customized configuration, run the `upgrade` command with the `--mode clean` parameter to delete the existing temporary directory and create a new one.
6. Open Merge Tracker. All the resolved files that changed in the customized configuration are automatically unresolved and tagged as **changed-after-resolution**.



ChargeHoldsPopup.pcf *NO_CHANGE NewBase*
ChargesLV.pcf *BOTH_EDIT NewBase (109/1)* [changed-after-resolution]
CommissionRatesPopup.pcf *GW_EDIT NewBase*
JournalScreen.pcf *NO_CHANGE NewBase*
LedgerScreen.pcf *GW_EDIT NewBase*
NewChargeReversalConfirmationPopup.pcf *NO_CHANGE NewBase*
TAccountOwnersLV.pcf *GW_EDIT NewBase*
TransactionDetailDV.pcf *GW_EDIT NewBase*
TransactionsLV.pcf *GW_EDIT NewBase*
TroubleTicketTransactionsLV.pcf *NO_CHANGE NewBase*

7. If you want to merge a file with the **changed-after-resolution** tag, see “Merging and resolving modified files from Merge Tracker” on page 69.
8. If you want to get more information about the changes in the temporary directory, run the following Git commands.

   `git status`

   Lists all changed files.

   ```
   On branch master
   Changes not staged for commit:
     (use "git add <file>..." to update what will be committed)
     (use "git checkout -- <file>..." to discard changes in working directory)

           modified:  tmp/cfg-upgrade/modules/configuration/config/web/pcf/accounting/ChargesLV.pcf
   ```

   `git diff`

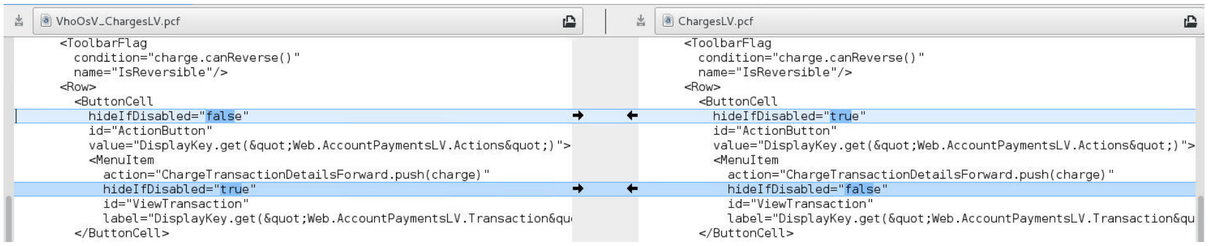
   Shows changes made to files.

   ```
   --- a/tmp/cfg-upgrade/modules/configuration/config/web/pcf/accounting/ChargesLV.pcf
   +++ b/tmp/cfg-upgrade/modules/configuration/config/web/pcf/accounting/ChargesLV.pcf
   @@ -80,12 +80,12 @@
           name="IsReversible"/>
   ```

```
        <Row>
          <ButtonCell
            hideIfDisabled="false"
            hideIfDisabled="true"
            id="ActionButton"
            value="DisplayKey.get(&quot;Web.AccountPaymentsLV.Actions&quot;)">
            <MenuItem
              action="ChargeT ransactionDetailsForward.push(charge)"
              hideIfDisabled="true"
              hideIfDisabled="false"
              id="ViewTransaction"
              label="DisplayKey.get(&quot;Web.AccountPaymentsLV.Transaction&quot;)"/>
          </ButtonCell>
```

git difftool

Shows changes made to files in the default diff tool from the Git configuration.



VhoOsV_ChargesLV.pcf

```
      <ToolbarFlag
        condition="charge.canReverse()"
        name="IsReversible"/>
      <Row>
        <ButtonCell
          hideIfDisabled="false"
          id="ActionButton"
          value="DisplayKey.get(&quot;Web.AccountPaymentsLV.Actions&quot;)">
          <MenuItem
            action="ChargeTransactionDetailsForward.push(charge)"
            hideIfDisabled="true"
            id="ViewTransaction"
            label="DisplayKey.get(&quot;Web.AccountPaymentsLV.Transaction&qu
        </ButtonCell>
```

ChargesLV.pcf

```
      <ToolbarFlag
        condition="charge.canReverse()"
        name="IsReversible"/>
      <Row>
        <ButtonCell
          hideIfDisabled="true"
          id="ActionButton"
          value="DisplayKey.get(&quot;Web.AccountPaymentsLV.Actions&quot;)">
          <MenuItem
            action="ChargeTransactionDetailsForward.push(charge)"
            hideIfDisabled="false"
            id="ViewTransaction"
            label="DisplayKey.get(&quot;Web.AccountPaymentsLV.Transaction&qu
        </ButtonCell>
```

chapter 5

Download and install the InsuranceSuite Configuration Upgrade Tools

Before you begin

The InsuranceSuite Configuration Upgrade Tools 3.1.0 require:

- Java Development Kit 11
- An InsuranceSuite application with Guidewire Studio 5.x.x.

About this task

The InsuranceSuite Configuration Upgrade Tools 3.1.0 are not included in your InsuranceSuite application. You can get them from the Guidewire Community.

After you install the tools, you must create an `upgrade.properties` file from the supplied `upgrade.properties.template` file. The `upgrade.properties` file stores various parameters for the InsuranceSuite Configuration Upgrade Tools, such as the location of the configuration that you want to upgrade.

Download the InsuranceSuite Configuration Upgrade Tools

Procedure

1. Visit the Guidewire Community:
   - **Guidewire Customers** - `https://community.guidewire.com`
   - **Guidewire Partners** - `https://partner.guidewire.com`
2. Select **Resources→Product Releases**.
3. Under **Product Group**, select **InsuranceSuite**.
4. Under **Product**, select **InsuranceSuite Configuration Upgrade Tools**.
5. Under **Release**, select **3.1.0**.
6. Download the software by clicking the download link.

Install the InsuranceSuite Configuration Upgrade Tools

Procedure

1. Quit Guidewire Studio if it is running.
2. If the `upgrade/etc/upgrade.properties` file already exists in the installation directory of the target InsuranceSuite application, save a copy of the file and then delete the `upgrade` directory.
3. Extract the `upgrade` directory from the downloaded `InsuranceSuite_Configuration_Upgrade_Tools_300.zip` file into the installation directory of the target InsuranceSuite application.
4. Verify that the extracted `upgrade` directory contains the following subdirectories:
   - `bin`
   - `etc`
   - `merge-tracker`
   - `preupgrade`
   - `smartdiff`
   - `smartmerge`
   - `studio-5.x`
   - `upgrade`
   - `upgrade-inspections`

Create upgrade properties

About this task

When configuring upgrade properties, follow these formatting rules:

- Do not use quotes for paths, even if the paths include spaces.
- Guidewire recommends that you use forward slashes as path separators on all platforms. If you prefer to use Windows-style paths with backslashes as separators, the backslashes must be escaped. For example, both of the following paths are valid:
  - `C:/Tools/Apache/ant/apache-ant-1.7.1`
  - `C:\\Tools\\Apache\\ant\\apache-ant-1.7.1`

Procedure

1. Go to `upgrade/etc` in the installation directory of the target InsuranceSuite application.
2. Create a file named `upgrade.properties` by duplicating `upgrade.properties.template`.
3. Open `upgrade.properties` and set the following properties (other properties are not needed).

   Note: If you have a saved copy of the `upgrade.properties` file from a previous installation, you can use it for reference while configuring the upgrade properties.

   `upgrader.priorversion.dir`

   Path to the configuration that you want to upgrade. If you perform a full upgrade, use the path to the top-level directory of the source environment (for example, `C:/InstallDir_9`). If you perform a partial upgrade, use the path to the specific module (for example, `C:/MyAccelerator/configuration/`).

   `upgrader.exclude`

   Patterns for files and directories in the customized configuration that the upgrade steps omit during upgrade. Guidewire recommends excluding all files that are not the core part of the customized configuration, like temporary and generated files. Files in `modules/configuration/config/metadata/entity` and `modules/configuration/config/metadata/typelist` are automatically excluded from upgrade because they are read-only in Guidewire Studio and must not be customized.

Patterns must follow the syntax supported by the `getPathMatcher` method of the `FileSystem` class. For information on the syntax rules, see `java.nio.file.FileSystem#getPathMatcher(java.lang.String)` in the Java Platform documentation.

`upgrader.sourceant.home, upgrader.sourcejava.home`

Paths to the top-level directory of the Ant installation and the JDK installation used by the configuration that you want to upgrade. For example, if you upgrade from 8.x.x to 10.x.x, provide the paths to the Ant and Java versions used by 8.x.x, NOT the versions used by 10.x.x.

Note: You need the Ant and JDK parameters only for an upgrade from release 8 or earlier to be able to run preupgrade. The JDK version provided here is not the version required by the InsuranceSuite Configuration Upgrade Tools.

chapter 6

Extending the InsuranceSuite Configuration Upgrade Tools

The InsuranceSuite Configuration Upgrade Tools include an upgrade automation framework, which allows you to extend their functionality by adding custom steps.

The main advantage of the framework is the possibility to add automation when needed, regardless of the release schedule of the InsuranceSuite Configuration Upgrade Tools.

> **IMPORTANT** Make sure to test thoroughly any extension that you develop for the InsuranceSuite Configuration Upgrade Tools before you run it in your production environment.

Add or override an upgrade step

About this task

You can extend the functionality of the InsuranceSuite Configuration Upgrade Tools in two ways:

- Add your own upgrade step
- Override a default upgrade step delivered by Guidewire

Each upgrade step has its own YAML file with configuration properties. Configuration files for default upgrade steps delivered by Guidewire are stored in `upgrade/upgrade/etc/steps/guidewire`. Configuration files for custom upgrade steps are stored in `upgrade/upgrade/etc/steps/custom`.

When you add an upgrade step, in fact you create a new configuration for a template upgrade step that is built into the InsuranceSuite Configuration Upgrade Tools. The following template upgrade steps are available:

Copy files from resources to customer configuration

This step copies files, such as enhancements, from the resources to the upgraded customized configuration in the temporary directory.

Transform document

This step can transform files in the following ways:

- Modify XML files using the XPath query language
- Find and replace strings using Java regular expressions
- Check if the file content contains a Java regular expression and continue or terminate processing accordingly

- Rename, move, delete files

For more information on properties of the template upgrade steps, see “Upgrade steps properties” on page 27.

When you override a default upgrade step, you create a new configuration for the step that takes precedence over the existing configuration. Before running the upgrade process, the InsuranceSuite Configuration Upgrade Tools check if the directory with custom steps and the directory with default steps contain any files with the same name. If they find any duplicate files, they use the custom configuration file.

Add a new upgrade step

About this task

To add a new step, you create a YAML file with a unique name in the directory for storing custom upgrade steps.

Procedure

1. Go to `upgrade/upgrade/etc/steps/custom`.
2. Create a YAML file and name it *`custom-name`*`.step.yaml`.

   The step name is generated from the name of the YAML file by cutting out the `.step.yaml` part , capitalizing the first character in each segment and removing hyphens. For example, step names are used in the list of available upgrade steps that you can display using the `upgrade --list-steps` command.

   > **IMPORTANT** Make sure that the name of your file is different from the name of any default file in `upgrade/upgrade/etc/steps/guidewire`. Otherwise, your new step will override the default step.

3. Open the YAML file and add step properties as described in “Upgrade steps properties” on page 27.

Example

Create `delete-example-e-n-files.step.yaml` with the following configuration in `upgrade/upgrade/etc/steps/custom`.

```
description: "Delete example EN files"
targetVersion: 10.0.0
fileSets:
  - .*/config/extensions/typelist/.*\\.example.ttx
transformations:
  - [deleteFile]
```

This configuration file adds an upgrade step named **DeleteExampleENFiles**, which deletes all `.example.ttx` files in the `/config/extensions/typelist` directory. It runs for all InsuranceSuite applications when you upgrade from a release lower than 10.0.0 to a release that is equal to or higher than 10.0.0.

Next steps

Complete the following task: “Test an upgrade step” on page 25.

Override a default upgrade step

About this task

To override a default upgrade step, you create a YAML file in the directory for storing custom upgrade steps. The file must have the exact same name as the configuration file for the default step you want to override.

Procedure

1. Go to `upgrade/upgrade/etc/steps/custom`.

2. Create a YAML file that has the same name as the YAML file in `upgrade/upgrade/etc/steps/guidewire` that you want to override.

   IMPORTANT Changes to the default YAML files in the `upgrade/upgrade/etc/steps/guidewire` directory are not supported.

3. Open the file and add step properties as described in “Upgrade steps properties” on page 27.

Example

The default `territory-code.step.yaml` file in `upgrade/upgrade/etc/steps/guidewire` contains the following configuration.

```
description:
  > Convert 'TerritoryCode.createSearchCriteria' calls to 'TerritoryCode.createLookupCriteria' calls.
product: PC
targetVersion: 9.0.0
fileSets:
  - PCFS
  - GOSU
transformations:
- [replaceAll ,  "([cC]ode \\ .)createSearchCriteria \\ ( \\ )" ,  "$1createLookupCriteria()"]
```

The step performs a `replaceAll` transformation on PCF and Gosu files. It runs only for PolicyCenter when you upgrade from a release lower than 9.0.0 to a release that is equal to or higher than 9.0.0.

If you want this upgrade step to perform the transformation on product model files as well, you need to override the step by creating `territory-code.step.yaml` with `PRODUCT_MODEL` added to the `fileSets` property in `upgrade/upgrade/etc/steps/custom`.

```
description:
  > Convert 'TerritoryCode.createSearchCriteria' calls to 'TerritoryCode.createLookupCriteria' calls.
product: PC
targetVersion: 9.0.0
fileSets:
  - PCFS
  - GOSU
  - PRODUCT_MODEL
transformations:
- [replaceAll ,  "([cC]ode \\ .)createSearchCriteria \\ ( \\ )" ,  "$1createLookupCriteria()"]
```

Next steps

Complete the following task: “Test an upgrade step” on page 25.

Test an upgrade step

Before you begin

Back up the files that you will use for testing.

About this task

After you add an upgrade step or override a default step, you can test if your new configuration works as expected. Here are example scenarios for testing upgrade steps.

Test an upgrade step with full upgrade

About this task

This procedure shows how you can test an upgrade step during the regular configuration upgrade process.

Procedure

1. Run the following command.

   IMPORTANT All changes in the temporary directory that you have not merged will be lost.

   ```
   upgrade --step <stepName> --show-diffs --mode clean
   ```

   This command runs the upgrade step with parameters to delete the temporary directory and create a new one (`--mode clean`) and to log differences between source files and upgraded files in the temporary directory (`--show-diffs`).

   For example, if you want to test a step named **FixConflicts** use the following command.

   ```
   upgrade --step FixConflicts --show-diffs --mode clean
   ```

2. Analyze the command output to make sure the step performed correct transformations.

Test an upgrade step with partial upgrade

About this task

This procedure shows how you can test an upgrade step outside the regular configuration upgrade process.

Procedure

1. Run the following command.

   ```
   upgrade --partial <dirToUpgrade> --from <sourceAppAndVersion> --to <targetAppAndVersion> --show-diffs --
   step <stepName>
   ```

   This command runs the upgrade step with parameters to upgrade a specific directory (`--partial`) and to log differences between source files and upgraded files in the temporary directory (`--show-diffs`).

   IMPORTANT If the configuration of an upgrade step has the `fileSets` property defined and the structure of the directory that you upgrade does not match the regular expression used in this property, the upgrade step is not executed. In this case, you can use the `--ignore-file-set-restrictions` parameter to ignore the `fileSets` property and run the steps on all files in the directory.

   For example, if you want to test a step named **FixConflicts** on a set of files in the `TestFiles` directory for upgrade of PolicyCenter from release 8.0.0 to 10.0.2, use the following command.

   ```
   upgrade --partial /TestFiles --from pc-8.0.0 --to pc-10.0.2 --show-diffs --step FixConflicts
   ```

2. Analyze the command output to make sure the step performed correct transformations.

Listing available upgrade steps

You can display details of all available upgrade steps with the following command.

```
upgrade --list-steps
```

The following information is shown for each step:

Step name

The name generated from the name of the step configuration file by cutting out the `.step.yaml` part , capitalizing the first character in each segment and removing hyphens. For example, the step name generated from the `rename-u-s-english-properties.step.yaml` file is **RenameUSEnglishProperties**.

Step description

The value of the `description` property in the step configuration.

Step type

The step can have one of these types:

- **Guidewire** - a default step delivered by Guidewire (without steps that were overridden). Configuration files for the default steps are stored in `upgrade/upgrade/etc/steps/guidewire`.
- **Custom** - a step added to the InsuranceSuite Configuration Upgrade Tools by the customer. Configuration files for these steps are stored in `upgrade/upgrade/etc/steps/custom`.
- **Overriding** - a step added to the InsuranceSuite Configuration Upgrade Tools by the customer that overrides the configuration of the default step with the same name. Configuration files for these steps are stored in `upgrade/upgrade/etc/steps/custom`.

For more information on adding and overriding upgrade steps, see “Add or override an upgrade step” on page 23.

InsuranceSuite application

The value of the `product` property in the step configuration. If the property is not provided, **PL** is shown.

Target version

The value of the `targetVersion` property in the step configuration. If the property is not provided, ***.*.*** is shown.

For more information on properties of the upgrade steps, see “Upgrade steps properties” on page 27.

Upgrade steps properties

This topic describes properties that you can use when you add a new upgrade step and override a default step.

Name	Description	Value	Required/ optional
`class`	The Java class that defines the template upgrade step. If this property is not provided, the **Transform document** step is used.	• Fully qualified name of a Java class • Example: `class: com.guidewire.upgrade.steps.general.pl.TransformDocumentUpgradeStep` Accepted names for new upgrade steps: • For **Transform document**: `com.guidewire.upgrade.steps.general.pl.TransformDocumentUpgradeStep` • For **Copy files from resources to customer configuration**: `com.guidewire.upgrade.steps.general.pl.CopyFilesFromResourcesToCustomerConfigurationUpgradeStep` **Note:** Guidewire recommends that you leave the `class` property unchanged in overriding steps.	Optional
`description`	Information on how the upgrade step works	• String • Markdown syntax is accepted. It may be useful in a situation where you redirect the output of the `upgrade --list-steps` command to a file and then display it with a tool that parses Markdown. • Example: `description: Delete **example** files`	Required
`product`	The InsuranceSuite application for which the upgrade step runs.	• Application code • Example: `product: PC`	Optional

Name	Description	Value	Required/ optional
	If this property is not provided, the upgrade step runs for all InsuranceSuite applications.	Accepted application codes: • `AB` for ContactManager • `BC` for BillingCenter • `CC` for ClaimCenter • `PC` for PolicyCenter	
`targetVersion`	The release ID of the InsuranceSuite application for which the upgrade step runs. The step runs if: • The release ID of the current InsuranceSuite application is lower than `targetVersion` and the release ID of the new InsuranceSuite application is equal to or higher than `targetVersion`. • The release ID of the current InsuranceSuite application and the release ID of the new InsuranceSuite application are both equal to `targetVersion`. For example, if an upgrade step has `targetVersion` set to `10.0.0` in the configuration: • It runs when you upgrade from 9.x or 10.0.0 to 10.0.0. • It does not run when you upgrade from 10.0.0 to 10.0.3. If this property and `targetVersionUntil` are not provided, the step runs for all releases.	• String in the following format: `<major>.<minor>.<maintenance>` • You can get the release ID of an InsuranceSuite application from information in `modules/configuration/product.properties`. • Example: `targetVersion: 9.0.0`	Optional
`targetVersionUntil`	The release ID of the InsuranceSuite application for which the upgrade step runs. The step runs if the release ID of the new InsuranceSuite application is lower than `targetVersionUntil`. If this property and `targetVersion` are not provided, the step runs for all releases.	• String in the following format: `<major>.<minor>.<maintenance>` • You can get the release ID of an InsuranceSuite application from information in `modules/configuration/product.properties`. • Example: `targetVersionUntil: 10.0.1`	Optional
`enabled`	Allows you to enable and disable the upgrade step. If this property is not provided, the step is enabled. By default, disabled steps do not run when you use the `upgrade` command. To run a disabled step, use the `upgrade` command with the `--enable-step <STEP>` option.	• Boolean • Example: `enabled: false`	Optional

Copy files from resources to customer configuration

The table contains properties of the **Copy files from resources to customer configuration** template step. You select the template step through the `class` property.

Name	Description	Value	Required/ optional

`copyResources`	A map of key-value pairs defining the resources path and the target path	• Key-value pairs (*`<resourcesPath>`* `:` *`<targetPath>`*)
		• The resources directory used by default upgrade steps is not available in the InsuranceSuite Configuration Upgrade Tools package. For custom upgrade steps, use a resources path relative to `upgrade/upgrade/etc`.
		• If the target path is empty, the default path is used: `gsrc/gw/api/upgrade/enhancements`
		• Example (the resources directory is `upgrade/upgrade/etc/resources`):
		`copyResources:`
		`  "resources/MyEnhancement.gsx" :`
		`"gsrc/gw/api/upgrade/enhancements"`
			Required

Transform document

The table contains properties of the **Transform document** template step. You select the template step through the `class` property.

Name	Description	Value	Required/ optional

`fileSets`	A list defining files transformed by the upgrade step	• List of file types and/or regular expressions supported by Java
		• Example:
		`fileSets:`
		`  - PCFS`
		`  - GOSU`
		`  - .*/config/resources/productmodel/`
		`auditschedules.*\.xml`
		Accepted file types:
		• `DISPLAY_NAMES` for `.en` files
		• `ENTITIES` for `.eix`, `.eti` and `.etx` files in `extensions`
		• `GOSU` for `.gs`, `.gsp`, and `.gsx` files
		• `LOCALE_PROPERTIES` for `.properties` files in `locale`
		• `LOOKUPTABLES` for `.xml` files in `lookuptables`
		• `PCFS` for `.pcf` files
		• `PLUGINS` for `.gwp` files
		• `PRODUCT_MODEL` for `.xml` and `-lookups.xml` files in `productmodel`
		• `RULES` for `.gr` files
		• `SYSTABLES` for `.xml` files in `systables`
		• `TYPELISTS` for `.tix`, `.tti` and `.ttx` in `extensions`
		• `WORKFLOWS` for `.xml` files in `workflow`
			Required

<table>
<thead>
<tr><th>Name</th><th>Description</th><th>Value</th><th>Required/ optional</th></tr>
</thead>
<tbody>
<tr>
<td><code>transformations</code></td>
<td>A list defining transformation types and their arguments</td>
<td>
<ul>
<li>List</li>
<li>Syntax: <i><code>&lt;transformationName&gt;</code></i>, <i><code>&lt;arg1&gt;</code></i>, <i><code>&lt;arg2&gt;</code></i>, ...</li>
<li>Example:
<pre>
transformations:
  - [setAttribute , "(//*:edgeForeignKey
| //*:edgeForeignKey-override)
[*:implementsEntity[@name =
          'Extractable']]" ,
extractable , "'true'"]
  - [setAttribute , "(//*:edgeForeignKey
| //*:edgeForeignKey-override)
[*:implementsEntity[@name =
          'OverlapTable']]" ,
overlapTable , "'true'"]
  - [removeElement , "(//*:edgeForeignKey |
          //*:edgeForeignKey-override)/
*:implementsEntity[@name='Extractable' or
          @name='OverlapTable']"]
</pre>
</li>
</ul>
Transformations:
<ul>
<li><code>setAttribute</code>
<ul>
<li>Arguments: <code>"<i>&lt;XPathExpressionToFindElement&gt;</i>", "<i>&lt;attributeNameToSet&gt;</i>", "<i>&lt;relativeXPathExpression&gt;</i>"</code></li>
<li>Example: <code>[setAttribute, "/Entity/Columns/DisplayNameColumn[not(@displayName)]", "displayName", "'true'"]</code></li>
</ul>
</li>
<li><code>removeAttribute</code>
<ul>
<li>Arguments: <code>"<i>&lt;XPathExpressionToFindElement&gt;</i>", "<i>&lt;attributeNameToDelete&gt;</i>"</code></li>
<li>Example: <code>[removeAttribute, "(//*:foreignkey|//*:edgeForeignKey)[@ondelete]", "ondelete"]</code></li>
</ul>
</li>
<li><code>insertChildElement</code>
<ul>
<li>Arguments: <code>"<i>&lt;XPathToFindTheParentElement&gt;</i>", "<i>&lt;relativeXPathExpression&gt;</i>"</code></li>
<li>Example: <code>[insertChildElement, "//*:typecode[not(*[@code='UWRule'])]", "'<category code=\'UWRule\' typelist=\'Rule\' /&gt;'"]</code></li>
</ul>
</li>
<li><code>moveElementBefore</code>
<ul>
<li>Arguments: <code>"<i>&lt;XPathExpressionToFindElement&gt;</i>", "<i>&lt;relativeXPathExpression&gt;</i>"</code></li>
<li>Example: <code>[moveElementBefore, "//*:PanelRef[@def]/(CardViewPanel|PanelSet|SearchPanel|TemplatePanel|TreeViewPanel)", ".."]</code></li>
</ul>
</li>
<li><code>renameElement</code>
<ul>
<li>Arguments: <code>"<i>&lt;XPathExpressionToFindElement&gt;</i>", "<i>&lt;targetElementName&gt;</i>"</code></li>
<li>Example: <code>[renameElement, "/Entity/Columns/DisplayNameColumn", "Column"]</code></li>
</ul>
</li>
</ul>
</td>
<td>Required</td>
</tr>
</tbody>
</table>

Name	Description	Value	Required/ optional
		• `removeElement` ◦ Arguments: `"<XPathExpressionToFindElement>"` ◦ Example: `[removeElement, "/*/column[../implementsEntity[@name = 'AddressBookLinkable'] and @name = 'AddressBookUID']"]` • `replaceAll` ◦ Arguments: `"<regularExpressionPattern>", "<replacement>"` ◦ Regular expression must be supported by Java ◦ Example: `[replaceAll, "EffDated\\s+&\\s+ScheduledItem", "ScheduledItem"]` • `continueIfMatches` ◦ Any subsequent transformation is executed only if the file content contains the specified regular expression ◦ Arguments: `"<regularExpressionPattern>"` ◦ Regular expression must be supported by Java ◦ Example: `[continueIfMatches, "(implements\\s+CoverableAdapter)"]` • `deleteFile` ◦ Arguments: None ◦ Example: `[deleteFile]` • `renameFile` ◦ Renames or moves the file ◦ Arguments: `"<regularExpressionPattern>", "<replacement>"` ◦ Regular expression must be supported by Java ◦ Example: `[renameFile, "Popup.pcf$", "600Popup.pcf"]` **Note:** If an XML file that you want to transform includes a namespace declaration (`xmlns`), use `*:` before the element name in the XPath expression to look for the element in all namespaces. For example, `/*/*:edgeForeignKey` or `/*/*:edgeForeignKey/*:implementsEntity[@name='Extractable']`.	

chapter 7

Preupgrade tool

The preupgrade tool prepares a 7.x or 8.x configuration for the automated upgrade process by performing a number of code cleanup and formatting steps. Changes made by the preupgrade tool are destructive and cannot be rolled back because the tool modifies your original customized configuration. For example, if you are upgrading an InsuranceSuite application from release 8 to 10, the preupgrade tool will modify the configuration of the InsuranceSuite application 8.0.

Run preupgrade steps

Before you begin

Verify the configuration of the InsuranceSuite application to be upgraded:

1. Ensure that your customized configuration has no compilation errors. Such errors can lead to errors in `server.log` when the preupgrade steps execute.
2. Ensure that your customized configuration can be started successfully.

About this task

IMPORTANT The preupgrade process is destructive and cannot be rolled back. It is therefore vital that you do not upgrade directly from the original installation. Instead, make a copy of the original installation and upgrade from that working copy (which you can delete when the upgrade is complete).

Procedure

1. Open a new command window and set up a development environment for the target release.
2. Navigate to the `upgrade/bin` directory in the target environment.
3. Run the following command:

   ```
   preupgrade > preupgrade.log
   ```

   The preupgrade tool will create the following files.

   In the source environment:

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

   In the target environment:

   - `upgrade/bin/preupgrade.log`

4. Verify that the preupgrade ran to completion:
   - The `upgrade/bin/preupgrade.log` file must exist in the target environment. Check this log file for errors and warnings. Verify that the preupgrade process succeeded by finding a line that indicates the process is complete.

     For example:

     ```
     1/6/17 6:55:28 AM EST: Preupgrade complete, 8,556 total files upgraded
     ```
   - The `tmp/src-upgrade/server.log` file must exist in the source environment. Check the log file for errors and warnings. During preupgrade, the server is shut down before it completes start-up. This is expected behavior. If you see other errors, address them before continuing.

Preupgrade command line options

To start the preupgrade tool from the command line, run `preupgrade` from the `upgrade/bin` directory.

The following tables list preupgrade options.

Basic options

To list the basic options, run `preupgrade --help`.

Option	Action
<PATH>	Path to the configuration that you want to upgrade. This parameter overrides the configuration path from the `upgrade.properties` file.
`--ant-home` *<PATH>*	Path to the top-level directory of the Ant installation associated with your customized configuration. This parameter overrides the Ant path from the `upgrade.properties` file. For example, for upgrade from 8.x.x to 9.x.x, the path must point to the Ant version compatible with 8.x.x, not the version compatible with 9.x.x.
`--dry-run`	Do a test run (simulate execution without saving results) (default: false)
`--java-home` *<PATH>*	Path to the top-level directory of the Java JDK installation associated with your customized configuration. This parameter overrides the Java path from the `upgrade.properties` file. For example, for upgrade from 8.x.x to 9.x.x, the path must point to the Java version compatible with 8.x.x, NOT the version compatible with 9.x.x.
`--max-mem` *<AMOUNT>*	Maximum amount of memory that the server can use (in megabytes) (default: 8000)
`--max-perm` *<AMOUNT>*	Maximum amount of memory that the server JVM `PermGen` space can use (in megabytes) (default: 512)
`--server-log` *<PATH>*	Pathname to save the server log file, relative to the home directory of the Guidewire product configuration (for example, `logging/myServer.log`). Use - to print to console without creating a log file. (default: `tmp/src-upgrade/server.log`)
`-X`	Print help including advanced and internal options (default: false)
`-h (--help)`	Print help (default: true)

Advanced options

To list the advanced options together with the basic options, run `preupgrade -X`.

Option	Action
`--clean-up`	Clean up leftover server settings related to preupgrade (default: false)
`--debug` *<PORT>*	Enable debugging on a specific port. By default, debugging is disabled.
`--files` *<PATH>*	Preupgrade only files in specific locations. Provide a list of absolute file paths.
`--group-steps`	Run preupgrade steps in groups (default: true)
`--mode [CUSTOMER | BASE | BOTH]`	Select configuration to preupgrade: • `BASE` - base configuration • `CUSTOMER` - customized configuration • `BOTH` - base and customized configurations (default: `BOTH`)
`--names` *<NAME>*	Preupgrade only specific types. Provide a list of fully qualified type names.
`--normalize`	Normalize source code to reduce insignificant changes without running preupgrade steps (default: false)
`--optional-kinds`	Preupgrade also optional Gosu source kinds (`DocumentTemplate` and `DocumentTemplateDescriptor`) (default: false)
`--remove-lock-files`	Remove leftover lock files from the snapshot repository (default: false)
`--skip-kind [EntityName | GosuProgram | GosuTemplate | GosuClass | PCF | Workflow | ProductModel | DocumentTemplate | DocumentTemplateDescriptor | GX | DiffTree]`	Gosu source kind to skip. Use this option multiple times with different kind names to skip multiple kinds.
`--skip-step` *<STEP>* `--step` *<STEP>*	Name of the preupgrade step to skip or run. Use this option multiple times with different step names to skip or run multiple steps. For more information about the steps, see "Preupgrade steps reference" on page 35.
`--timeout` *<TIMEOUT>*	Time limit for running a single preupgrade step (in seconds). After this limit is exceeded, an exception is logged for the step and the preupgrade process continues. (default: 600)
`--unused-ports`	Configure the server to use free ports (default: false)
`--warnings-as-errors`	Save warnings produced by preupgrade steps as errors in the error log file (default: false)

Preupgrade steps reference

This topic lists steps that are run by preupgrade.

BuiltinTypes

Replaces certain aliases for types with their real names.

For example, `Number` becomes `java.lang.Double`.

CaseSensitivity

Corrects the case of all references.

ClaimSnapshot

Replaces `snapshot.*` types with `dynamic.Dynamic`. This step is only for ClaimCenter.

CoercionsRemoval

Removes coercions which are not compatible with Gosu in 9.0.

CollectionsSignatureChange

Runs only for upgrades from a pre-8.0 version.
Preserves runtime semantics after signature changes to some collection methods between 7.0 and 8.0.
For example, `java.util.Map#get(T)` in 7.0 becomes `java.util.Map#get(Object)` in 8.0.

CurrentLocation

Converts calls to `CurrentLocation` methods to fully qualified calls.

DirectTypeAccessInfo

Replaces direct access of meta type features with `.Type`.
For example, `SomeType.isAssignableFrom(t)` becomes `SomeType.Type.isAssignableFrom(t)`.

DisplayKey

Replaces usages of display key types with Java API usage.
For example, `displaykey.Java.Text` becomes `DisplayKey.get("Java.Text")`.

EntityBeanPath

Fixes case sensitivity of entity bean paths.

EntityLoaders

Replaces entity literals with calls to `Queries.findByIdOrPublicId`.

ExceptUnless

Converts `unless` and `except` Gosu statements.

ExistsExpression

Replaces obsolete `exists` expressions with `HasMatch` equivalents.

ExpansionOperator

Replaces uses of the `.` operator as an expansion operator with explicit expansion operator `*.`.

ExternalEntityCoercion

Replaces implicit coercion for external entities with explicit conversion calls.

FindToQuery

Replaces obsolete `find` expressions with `Query.make` expressions.

FixInequalityOperator

Replaces the obsolete `<>` operator with the `!=` operator.

GxModelXmlTypeAttribute

Adds the xmlType attribute to include elements in GX Model files. The step does not make any changes to xmlType for byte[].

IntSubtypeTypekey

Replaces typekey.XxxIntSubtype with typekey.Xxx.

JavaStyleCast

Converts Java-style (Type) expr to Gosu-style expr as Type.

JavaStyleConstructor

Converts Java-style constructors of the form function <CLASS_NAME>() to Gosu-style construct().

KeywordsCaseSensitivity

Runs only for upgrades from a pre-8.0 version.

Fixes the case of all keywords.

For example, AND will become and.

Normalize

Normalizes source code to reduce semantically insignificant differences introduced by upgrade steps.

ObsoleteLoggingApi

Replaces obsolete Logging API with slf4j API.

PcfBooleanLiteralCase

Corrects case inconsistencies in simple boolean expressions (true or false) inside PCF files.

PersistenceKey

Replaces com.guidewire.commons.entity.Key with gw.pl.persistence.core.Key.

QueryBlockSyntax

Replaces obsolete query block expressions with IQueryResult.transformQueryRow calls.

QueryToQueryBeanResult

Replaces all references to entity.*Query types with gw.api.database.IQueryBeanResult<entity.*>.

RelativeImports

Removes use of the relative uses feature of Gosu from Gosu and PCF files.

For example, given a uses gw.api.* statement, references to admin.BaseAdminUtil become gw.api.admin.BaseAdminUtil.

SortByAttributeCaseSensitivity

Corrects the case of sortBy attributes in PCF row iterator cells.

UsesLocation

Moves uses statements from function bodies to the top of the file.

Wildcards

Removes usages of unsupported wildcard types from Gosu and PCF code.

XmlApi

Updates code which uses XML type loader to comply with API changes:

Adds `$` to `XmlElement` property names.

Replaces `XmlEnum.valueOf` calls with `XmlEnum.forValue` calls to preserve null-safe behavior.

XmlPackageCaseSensitivity

Converts XML package names to lower case.

chapter 8

Upgrade tool

The upgrade tool prepares your configuration for the manual merge process by performing a number of automated steps. Changes made by the upgrade tool are safe because it copies your original customized configuration to a temporary location and runs a series of automated upgrade steps on this temporary folder.

Run upgrade steps

About this task

The upgrade tool first copies the modules from the customized configuration to a `tmp/cfg-upgrade/modules` directory in the target environment. The location of the customized configuration is specified by the `upgrader.priorversion.dir` property in `upgrade/etc/upgrade.properties` in the target environment.

Then, the tool performs a number of automated steps to upgrade the copied customized configuration (in the `tmp/cfg-upgrade/modules` directory) to the target release of the InsuranceSuite Configuration Upgrade Tools. For detailed information on each step, see "Upgrade steps reference" on page 44.

Procedure

1. Open a command window and set up a development environment for the target release.
2. Navigate to the `upgrade/bin` directory in the target environment.
3. Run the following command:

   ```
   upgrade > upgrade.log
   ```

4. Check the upgrade log for errors.

Upgrade command line options

To start the upgrade tool from the command line, run `upgrade` from the `upgrade/bin` directory.

The following table lists upgrade options.

Option	Action
<PATH>	Path to the configuration that you want to upgrade. This parameter overrides the configuration path from the `upgrade.properties` file.

Option	Action
	If you perform a full upgrade, use the path to the top-level directory of the source environment (for example, `../../../InstallDir_9`). If you perform a partial upgrade, use the path to the specific module (for example, `MyAccelerator/modules/configuration/`).
`--enable-step` *`<STEP>`*	Name of the upgrade step disabled in the configuration file that you want to run. Use this option multiple times with different step names to enable multiple steps. To view the list of available upgrade steps, use the `--list-steps` option.
`--exclude` *`<PATTERN>`*	Patterns for files and directories in the customized configuration that the upgrade steps omit during upgrade. This parameter overrides the patterns from the `upgrade.properties` file. Patterns must follow the syntax supported by the `getPathMatcher` method of the `FileSystem` class. For information on the syntax rules, see `java.nio.file.FileSystem#getPathMatcher(java.lang.String)` in the Java Platform documentation. (default: `glob:{CVS,**/CVS,.svn,**/.svn,.git,**/.git,.idea,repository,*/*/target,*/*/generated,*/*/generated_classes,**/*.iml,tmp,build,admin,**/out}`)
`--from` *`<PRODUCT_AND_VERSION>`*	Name and version of the source application, for example pc-9.0.4. This argument is used together with `--partial`.
`--ignore-file-set-restrictions`	Ignore the `fileSets` property in the configuration of the upgrade steps and run the steps on all files in the directory. This argument is used together with `--partial`. (default: false)
`--list-steps`	Show details about available upgrade steps. The steps are listed in three groups: Guidewire (default steps delivered by Guidewire), Custom (steps added to the InsuranceSuite Configuration Upgrade Tools by the customer), Overriding (steps added to the InsuranceSuite Configuration Upgrade Tools by the customer that override the configuration of the default steps). (default: false)
`--mode [create | append | overwrite | clean | update]`	Select the upgrade tool mode: • `create` - create the temporary directory and copy all source files to it. Fail the upgrade if `tmp` already exists. • `append` - copy only the source files that do not exist in the temporary directory (changed and deleted files are not included). If `tmp` does not exist, create it and copy all source files to it. • `overwrite` - copy all source files to the temporary directory. Overwrite the files that already exist in `tmp`. If `tmp` does not exist, create it and copy all source files to it. • `clean` - create the temporary directory and copy all source files to it. If `tmp` already exists, delete it first. • `update` - run an upgrade directly on the temporary directory. Fail the upgrade if `tmp` does not exist. The mode does not affect which upgrade steps are run. (default: `create`)
`--partial`	Upgrade a specific part of the configuration. If you run this switch without the *`<PATH>`* parameter, the configuration path from the `upgrade.properties` file is used. This switch requires the `--from` and `--to` arguments that specify the name and version of the source application and the name and version of the target application. (default: false)

Option	Action
`--show-diffs`	Log differences between source files and upgraded files in the temporary directory. Transformations made by the **Reformat XML** step are not logged. (default: false)
`--skip-step` *<STEP>* `--step` *<STEP>*	Name of the upgrade step to skip or run. Use this option multiple times with different step names to skip or run multiple steps. To view the list of available upgrade steps, use the `--list-steps` option. For more information about the steps, see "Upgrade steps reference" on page 44.
`--to` *<PRODUCT_AND_VERSION>*	Name and version of the target application, for example pc-9.0.4. This argument is used together with `--partial`.
`-X`	Print help including advanced and internal options (default: false)
`-h (--help)`	Print help (default: true)
`-r (--dry-run)`	Do a test run (simulate execution without saving results) (default: false)
`-t (--target)` *<PATH>*	Path to the target environment. Use this parameter if you installed the InsuranceSuite Configuration Upgrade Tools outside the home directory of the new release of the InsuranceSuite application. (default: the directory where the InsuranceSuite Configuration Upgrade Tools are installed)

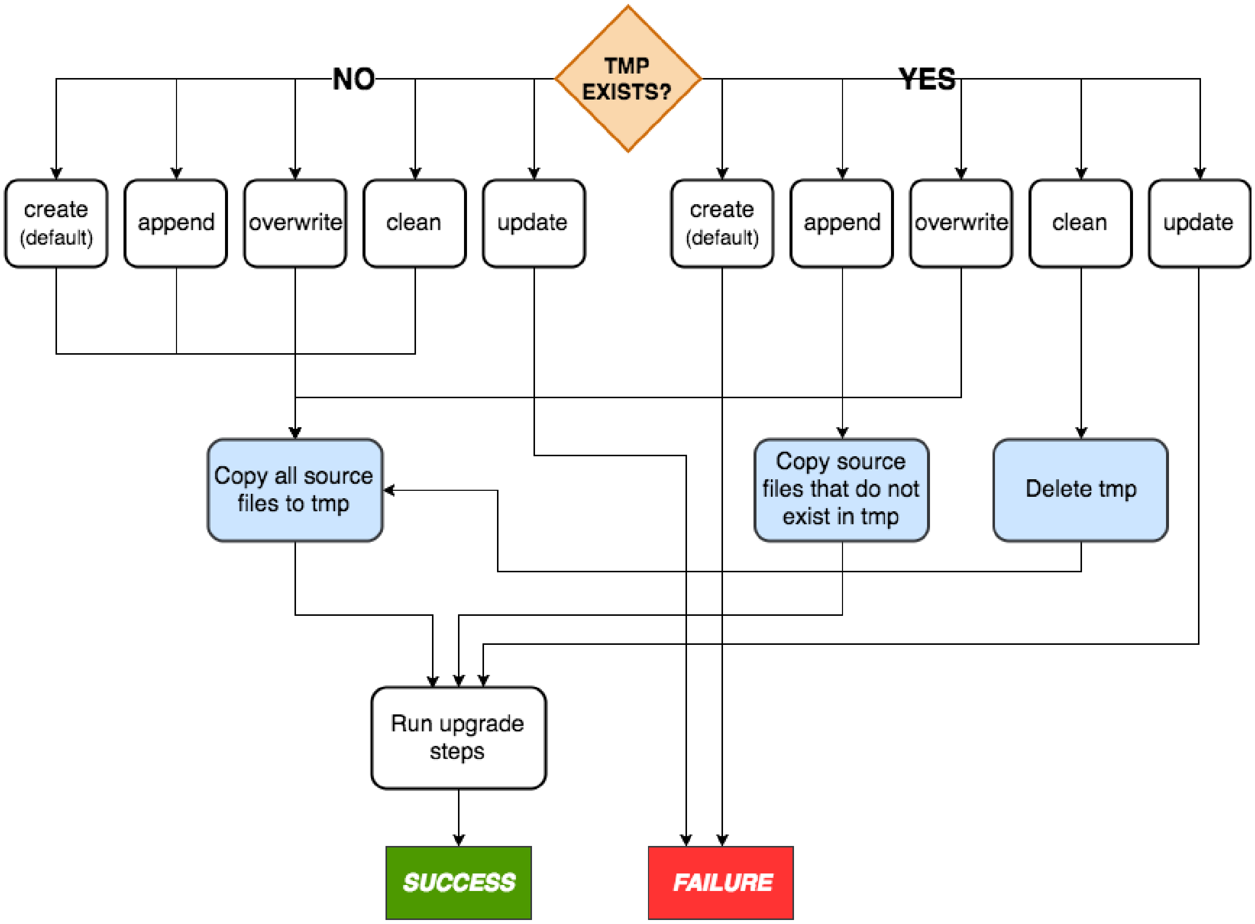
Upgrade tool modes

You can run the upgrade tool in several modes. By default, the upgrade tool runs in the `create` mode. To choose a different mode, use the `--mode` parameter.

```
upgrade --mode <mode name>
```

For more information on available modes, see "Upgrade command line options" on page 39.

Each mode has a different strategy for handling the temporary directory as shown in the following diagram.



Note: The mode does not affect which upgrade steps are run.

Example usage scenarios

Configuration upgrade is starting

If you run the upgrade tool for the first time, use the `create` mode. This mode is enabled by default so you do not need to use the `--mode` parameter.

Configuration upgrade is already in progress

> **Note:** Guidewire recommends that the **Unresolve file when changes in upgraded customized configuration are detected** option is enabled in the Merge Tracker settings. It allows you to keep your merging progress when you run the upgrade tool in a mode that modifies the temporary directory. For more information on this option, see “Configuring Merge Tracker” on page 74.

While performing an upgrade, you may need to run upgrade steps again for various reasons. The following examples help you decide which mode you need to use in a particular scenario:

- If files were added to the customized configuration, use the `append` mode to copy the new files to `tmp`.
- If some files were modified in the customized configuration, use the `ovewrite` mode to replace the previous version of the modified files in `tmp` with the new version.
- If some files were removed, moved or renamed in the customized configuration, use the `clean` mode to delete `tmp` and then create a new one. In a situation where you added the removed, moved or renamed files to the new customized configuration, use the upgrade inspection **Find customer files copied to the new customized configuration and then removed from the tmp directory**.

- If an upgrade step was added to the InsuranceSuite Configuration Upgrade Tools, use the `update` mode together with the `--step` parameter to run the new step on the existing `tmp`.
- If the behavior of an upgrade step was changed, use the `clean` mode to delete `tmp` and then create a new one. In a situation where an upgrade step was changed to copy different files from resources and you added the previously copied files to the new customized configuration, use the upgrade inspection **Find customer files copied to the new customized configuration and then removed from the tmp directory**.

Upgrade part of the configuration

About this task

You can use the InsuranceSuite Configuration Upgrade Tools to upgrade specific configuration components. All components that are not part of the base configuration, such as accelerators, are copied to the new customized configuration and marked as CUSTOMER_ADD. This is because the upgrade process does not have the base and new base versions of the file that it can use for comparison.

Use the --partial option

About this task

When you use the `upgrade` command with the `--partial` option, the upgrade process takes the configuration path from the `upgrade.properties` file, creates a `tmp/cfg-upgrade` directory in the directory where the InsuranceSuite Configuration Upgrade Tools are installed and puts all the upgraded files there.

Procedure

1. Go to the `upgrade/bin` directory.
2. Run the following command.

   ```
   upgrade --partial --from <PRODUCT_AND_VERSION> --to <PRODUCT_AND_VERSION>
   ```

   To override the configuration path, use the *`<PATH>`* parameter. For more information about command line options available for the upgrade tool, see “Upgrade command line options” on page 39.

   > **IMPORTANT** If the configuration of an upgrade step has the `fileSets` property defined and the structure of the directory that you upgrade does not match the regular expression used in this property, the upgrade step is not executed. In this case, you can use the `--ignore-file-set-restrictions` parameter to ignore the `fileSets` property and run the steps on all files in the directory.

Example

To upgrade a PolicyCenter module located in the `MyAccelerator` directory from version 9.0.6 to 10.1.0, run the following command.

```
upgrade ../../MyAccelerator --partial --from pc-9.0.6 --to pc-10.1.0
```

In this example, we use the *`<PATH>`* parameter to override the configuration path from the `upgrade.properties` file.

Upgrade to the same release

About this task

You can run a full upgrade when the source release and the target release of an InsuranceSuite application are the same. This feature allows you to upgrade specific configuration components to the latest release even if the InsuranceSuite release did not change.

Procedure

1. Make sure the source release and the target release of the InsuranceSuite application are the same.
2. Open `upgrade/etc/upgrade.properties`.
3. Set `upgrader.priorversion.dir` to the source environment path.
4. Go to `upgrade/bin`.
5. Run upgrade steps:

```
upgrade > upgrade.log
```

Upgrade steps reference

In InsuranceSuite release 9 and later, the tools are not included in InsuranceSuite applications. You can get the latest release of the InsuranceSuite Configuration Upgrade Tools and documentation from the Guidewire Community.

This topic lists default upgrade steps that are delivered by Guidewire in the InsuranceSuite Configuration Upgrade Tools 3.1.0.

The following table shows which steps run for specific upgrade paths.

Group of steps	Upgrade from 8.0	Upgrade from 9.0	Upgrade from 10.0.x
"9.0.0 upgrade steps" on page 44	✓		
"9.0.1 upgrade steps" on page 49	✓	✓ (only for upgrade from 9.0.0)	
"10.0.0 upgrade steps" on page 49	✓	✓	
"10.0.2 upgrade steps" on page 55	✓	✓	✓
"General upgrade steps" on page 55	✓	✓	✓

9.0.0 upgrade steps

These steps have `targetVersion` set to `9.0.0` and run when you upgrade from release 8.0.

Classes to adjust phone numbers

This step performs a series of actions related to properties of the `phone` type in the following order:

> **Note:** Actions 1-3 are performed only for custom entities in the customer configuration that do not have the `phonecountrycodeProperty` and `extensionProperty` columns.

1. Creates an enhancement with a set of properties for each entity that has columns of the `phone` type.
2. Creates `<entity>PhoneMappers.gs` for each entity that has at least one property of the `phone` type. The step also edits the implementation of the `EntityPhoneMapperEntries` property in `DefaultPhoneNormalizerPlugin.gs`.
3. Migrates data model elements of the `phone` type. The new step modifies entity files (ETI, EIX, ETX) and, if needed, adds extra columns: `phonecountrycodeProperty` and `extensionProperty`.
4. Creates a `PhoneParser.gs` class with an `exParsePhone` function and a `UniversalPhoneOwner.gs` class in the `gsrc/gw/api/phone/upgrade` directory.

Copy schemalocations to gwxmlmodule

This step copies the `schemlocations` element from `schemalocations.xml` to `gwxmlmodule.xml`.

Disabled PCF elements

This step converts commented out PCF nodes to disabled PCF nodes.

Fix broken Emerald trigger

This step fixes PCF `FormatCell` elements that have an error introduced by a 7.0 to 8.0 upgrade step.

This step that removes duplications inside `FormatCell` PCF elements. If the `FormatCell` is the only element in its `Row` then the step converts it back to a `DetailViewPanel` and moves it up the PCF hierarchy. In other cases it removes duplicate elements inside `FormatCell`.

Fix display key case sensitivity

This step changes display keys in `DisplayKey.get("displaykey")` expressions in Gosu and PCF files to be the same as their definitions in the `display.properties` or `display_*.properties file`. For example, `DisplayKey.get("pcf.Web.Errors.BillingSystem.PreventReactivationSyncFailed", period)` is changed to `DisplayKey.get("pcf.web.Errors.BillingSystem.PreventReactivationSyncFailed", period)`.

Flatten config locale

This step converts localization files directory to 9.0.x file structure.

Move abstract PanelDef outside of PanelRef

This step moves PCF `AbstractPanelDef` type elements outside of `PanelRef` elements in the PCF hierarchy, if the `PanelRef` element has a `def` attribute.

`AbstractPanelDef` elements include:

- `CardViewPanel`
- `ChartPanel`
- `DetailViewPanel`
- `ListDetailPanel`
- `ListViewPanel`
- `PanelRow`
- `PanelSet`
- `SearchPanel`
- `TemplatePanel`
- `TreeViewPanel`

PCF API

This step replaces `com.guidewire.pl.web.navigation` package references with `pcf.api` package references.

PCF relative packages

This step replaces relative package names with fully qualified package names in PCF files.

Reformat rules

This step reformats rule files to the 9.0.x rule format.

Remove AddressBookUID

This step removes the `AddressBookUID` definition from entities that implement the `AddressBookLinkable` delegate.

Remove autoupgrade attribute

This step removes the `autoupgrade` attribute from the `database` element in `database-config.xml`.

Remove DA params

This step removes Document Assistant related parameters from `config.xml`.

Remove DisplayNameColumn

This step replaces `DisplayNameColumn` PCF elements with `Column` PCF elements with attribute `displayName` set to `true`.

Remove JGroups params from config.xml

This step removes `JGroups` related parameters from `config.xml`.

Remove ondelete attribute

This step removes the non-functional `ondelete` attribute from entities.

Remove subpackage

This step removes the obsolete `subpackage` attribute from entities and typelists.

Remove unreachable rules

This step removes Gosu rules that are not listed in corresponding `order.txt` files.

By default, the step is disabled. To run it, use the `upgrade` command with the `--enable-step RemoveUnreachableRules` option. For more information, see “Upgrade command line options” on page 39.

Replace effdatedOnly

This step replaces the `effdatedOnly` attribute of `<delegate>` elements with `requiresType="effdated"` for the value.

Replace FK owner

This step makes the following changes:

- Replaces foreign key attribute `owner="true"` with `archivingOwner="source"`.
- Replaces foreign key attribute `owner="false"` with `archivingOwner="target"`.
- Adds foreign key attribute `archivingOwner="none"` when `<CrossDomain 0000000000000000PublicID>` is present.

Replace implementsEntity adapter

This step replaces the `adapter` attribute of `<implementsEntity>` elements with an `<implementsInterface>` child element. The new `implementsInterface` child element specifies the associated `requires` interface for the `iface` attribute, and specifies the value of the `adapter` attribute for the `impl` attribute.

Replace maxActive attribute

This step replaces `maxActive` attributes with `maxTotal` attributes in `dbcp-connection-pool` elements in `database-config.xml`.

Replace old display key calls

The step replaces usages of display key types with Java API calls. For example, `displaykey.Java.Text` becomes `DisplayKey.get("Java.Text")`. The step extends the functionality of the **DisplayKey** preupgrade step, which makes the same transformation but does not change nodes that are commented out.

Scriptability

This step converts unsupported getter and setter scriptability types `external`, `rules`, `ui`, and `workflow` to scriptability type `all`.

Update data types

This step fixes invalid data type names in entites and entity extensions.

Upgrade messaging-config.xml

This step adds the `defaultServer` attribute to `messaging-config.xml`.

Upgrade scheduler-config.xml

This step adds the `server` attribute to `scheduler-config.xml`.

Upgrade work-queue.xml

This step adds a `defaultServer` attribute to `work-queue.xml`.

ClaimCenter upgrade steps

These steps run only when you upgrade ClaimCenter configuration.

Rename 600 snapshot popups

This step renames old snapshot PCF files to include the version number.

PolicyCenter upgrade steps

These steps run only when you upgrade PolicyCenter configuration.

Add codeIdentifiers to product model

This step adds `codeIdentifier` attributes to product model elements.

Add valueType to product model

This step adds the valueType attribute to the DirectCovTermPattern, PackageTerm and OptionCovTermPattern elements in all the product model files (XML files in config/resources/productmodel) if the attribute is missing or empty. The valueType attribute is set to the following values:

- For DirectCovTermPattern: other
- For PackageTerm:
    - money if the value attribute contains "USD" or "$"
    - percent if the value attribute contains "%"
    - days if the value attribute contains "days"
    - hours if the value attribute contains "hours"
    - other if the value cannot be determined
- For OptionCovTermPattern:
    - money if the optionCode attribute of the CovTermOpt element (a child element of Options) contains "USD" or "$"
    - percent if the optionCode attribute of the CovTermOpt element (a child element of Options) contains "%"
    - days if the optionCode attribute of the CovTermOpt element (a child element of Options) contains "day"
    - hours if the optionCode attribute of the CovTermOpt element (a child element of Options) contains "hour"
    - other if the value cannot be determined

CovTermDirectInput

This step converts CovTermDirectInput PCF elements to InputSetRef PCF elements.

CovTermWidget

This step converts CovTermXxxInput and CovTermXxxCell PCF elements to RangeInput PCF elements.

CoverableAdapter

This step updates the CoverableAdapter and AbstractScheduledItemAdapter interfaces.

Product model

This step updates the product model directory structure within config/resources/productmodel and updates product model references in Gosu and PCF files.

- State-specific elements are moved to state-specific configuration files within state-specific subdirectories.
- All policy line patterns and associated lookup files are split according to their line of business (LOB) and moved into a LOB-specific *LOB_name* directory.
- All clause patterns and associated lookup files in the base PolicyCenter configuration are split up into LOB-specific *LOB_name*/coveragepatterns directories.
- All images are moved into LOB-specific *LOB_name*/images directories.
- Properties files are moved to a LOB-specific *LOB_name* directory.
- Clauses are directly linked to their associated policy line in the clause file. Coverage category information is still included but now becomes only an attribute of the clause, not the way it is linked to the policy line.
- Grandfathering information for offerings, modifiers, clauses, coverage terms and coverage term options is moved into separate files named *LOB_name*Line-grandfathering.
    - Base information is located in the *LOB_name*Line.xml file in the *LOB_name* directory.
    - Non-jurisdiction-specific grandfathering information is moved to a *LOB_name*Line-grandfathering in the *LOB_name* directory.
    - Jurisdiction-specific grandfathering information is moved to a *LOB_name*Line-grandfathering in the *LOB_name*/jurisdictions/*jurisdictionName* directory.

- State-specific and non-state-specific modifier minimum and maximum information are moved into state-specific and non-state-specific files.
- Product model lookup files for policy lines, clause, products, offerings and question sets are broken up into non-jurisdiction specific and jurisdiction-specific files and are placed in appropriate directories.
- Product pattern and associated lookup files in the base PolicyCenter configuration are broken up into product-specific `productName` directories.
- Offerings and associated look-up files in the base PolicyCenter configuration are broken up into product-specific `productName/offerings` directories.
- Product image files in the base PolicyCenter configuration are broken up into product-specific directories.

Rename AuditSchedulePatterns

This step renames the `AuditSchedulePattern` and `SingleCheckingAuditSchedulePattern` product model audit schedule patterns to either `SingleAuditSchedulePattern` or `SeriesAuditSchedulePattern`, depending on the `series` attribute, and remove the `series` attribute.

ScheduledItem

This step simplifies `ScheduledItem` multiple type expressions.

TerritoryCode

This step converts `TerritoryCode.createSearchCriteria` calls to `TerritoryCode.createLookupCriteria` calls.

WorksheetContainerAgeForPurging

This step renames parameter `WorksheetContainerAgeForPurging` to `RatingWorksheetContainerAgeForPurging` in config.xml.

9.0.1 upgrade steps

These steps have `targetVersion` set to `9.0.1` and run when you upgrade from releases:

- 8.0
- 9.0.0

Remove indexstatistics

This step removes the `indexstatistics` element from `database-config.xml`.

This change affects the following InsuranceSuite application functionality:

- The `system_tools -getdbstatisticsstatements` command no longer generates separate SQL statements for index statistics.
- The Server Tools **Database Catalog Statistics Information** download reports no longer contain statistics statements for indexes.

10.0.0 upgrade steps

These steps have `targetVersion` set to `10.0.0` and run when you upgrade from releases:

- 8.0
- 9.0

Add attribute desc to workflow files

The structure of workflow files changed to accommodate localization content for all languages within the same XML file.

This step adds the desc attribute to the AutoStep, ManualStep and Outcome elements in workflow files. The desc attribute is created from the id attribute. For example, desc="Wait Or Issue" is created from id="WaitOrIssue".

appendListViewPaging

PCF files allow multiple ListViews on the same page. The appendListViewPaging property of the TitleBar element was replaced with the appendPageInfo property of the RowIterator element.

If the ListView element and the TitleBar element are in the same PCF file, this step removes the appendListViewPaging property from the TitleBar element and sets the appendPageInfo property in the RowIterator element. If the ListView element is in a different PCF file and is modal, you must manually reconfigure the referenced ListView file.

BigDecimalUtil replace upgrade

This step replaces references to com.guidewire.util.BigDecimalUtil with references to gw.pl.util.BigDecimalUtil in Gosu files.

Change array casts to enhancement

This step checks if the AsArrayOfEnhancement.gsx file exists. If it exists, the step adds a Gosu enhancement file AsArrayOfEnhancementV10.gsx with the required reified modifier and comments out the content of AsArrayOfEnhancement.gsx.

Change structure of systables

The structure of system tables changed to accommodate localization content for all languages within the same XML file.

This step changes the structure of the system tables. It:

- Removes the <*_L10N_ARRAY> and <*>*some text in some language*</*> nodes
- Adds the <* language="en_US">*some text*</*> and <*language="*some_language_code*">some text in some language</*> nodes.

Equals replace upgrade

This step replaces lowercase usages of Relop operators within queries with their capitalized version in Gosu files.

Fix edgeForeignKey extractable or overlap

Guidewire has removed the implementsEntity element from edgeForeignKey and edgeForeignKey-override.

This step modifies entities in the following way:

1. Finds edgeForeignKey and edgeForeignKey-override elements that have a child implementsEntity element with the name attribute set to Extractable or OverlapTable.
2. Adds the extractable or overlapTable attribute to these elements and sets it to true.
3. Removes the child implementsEntity element.

getByCode replace upgrade

This step replaces product model pattern calls to XXPattern.getByCode() with calls to XXPatternLookup.getByCodeIdentifier() in PCF, Gosu, display names, and product model files.

Inbound integration config

This step converts inbound-integration-config.xml to the format compatible with release 10.x.

JurisdictionMappingUtil upgrade

The previously deprecated `JurisdictionMappingUtil.getJurisdictionMappingForPolicyLocation` method was removed.

This step replaces the `JurisdictionMappingUtil.getJurisdictionMappingForPolicyLocation` method with `JurisdictionMappingUtil.getJurisdiction` in PCF files, Gosu files and lookup tables.

PanelIteratorRemoveLink

In release 9, the `PanelIteratorRemoveLink` widget is used to remove panels from a `PanelIterator` element. A `PanelIteratorRemoveLink` widget can be placed anywhere in a panel as a child of a `PanelIterator` element.

In release 10, `PanelIterator` uses the same removal mechanism as `RowIterator`. Panels in a `PanelIterator` element automatically get a check box and a `RemoveButton` or `IteratorButtons` widget can be used to remove the selected panels. The `PanelIteratorRemoveLink` widget is no longer used.

This step removes the `PanelIteratorRemoveLink` widget from PCF files and adds a comment that you must reconfigure the file using an `IteratorButtons` or `RemoveButton` element.

PCF charts

This step performs a series of transformations on PCF files.

Height and width in ChartPanel

In release 9, the `width` and `height` properties for `ChartPanel` support expressions. In release 10, you can use these properties only with simple values, like 100, 50em and 20px. Values that are expressions can only be used in the `expressionHeight` and `expressionWidth` properties. However, these fields are marked as deprecated because Guidewire does not recommend using expressions for width and height.

The step transforms the `height` and `width` properties of `ChartPanel` that use expressions to `expressionHeight` and `expressionWidth`.

xValue for dual axis series

To make the configuration more consistent with other data series configurations, `xValue` must be used for dual axis series on continuous charts (`TimeSeries` and `XYLine`).

The step moves the value from `categoryLabel` to `xValue` for dual axis series on `TimeSeries` and `XYLine` charts.

HEX colors

In release 9, you can provide a HEX color code (for example, FF00FF) in the `rgb` attribute to configure a custom color for `DualAxisDataSeries` nodes. The provided color is used as a dual axis series color on the chart. In release 10, this attribute is no longer available. Instead, `fillColor` and `strokeColor` are used. To configure a custom color, you need to provide in both these attributes a Gosu expression that is evaluated to a `GWColor` instance.

The step replaces the `rgb` attribute and its HEX values with the `fillColor` and `strokeColor` attributes that use Gosu expressions to wrap the old HEX values, for example, `gw.api.web.color.GWColor.createFromHex("#FF00FF")`.

Chart type

The `type` property is no longer available in the `DomainAxis`, `RangeAxis` and `DualAxis` elements.

The step:

- Converts the chart's type from `XYLine` to `TimeSeries` if the `DomainAxis` element has the `type` attribute set to `Date`.
- Removes the `type` attribute from `DomainAxis`, `RangeAxis` and `DualAxis` elements.

Obsolete properties of charts and widgets

Chart implementation changed. The displaySectionLabels, innerPadding and disableAnimation properties of the ChartPanel element are obsolete now. ChartPanel may also have children of the DomainAxis or RangeAxis types which use the redundant lowerMargin property.

The step:

- Removes the displaySectionLabels, innerPadding and disableAnimation properties from ChartPanel.
- Removes the lowerMargin property from the children DomainAxis and RangeAxis elements of ChartPanel.

autoScale property

Charts scale automatically by default.

The step removes the autoScale property from charts.

categoryLabelOrientation property

The categoryLabelOrientation property of ChartPanel was renamed to labelOrientation because it works with other chart types, not just category-based charts. The property was also moved from ChartPanel to DomainAxis because it only impacts the labels on that axis.

The step renames the categoryLabelOrientation property of ChartPanel to labelOrientation and moves the property from ChartPanel to DomainAxis.

label property

You need to define the categoryLabel property on a DataSeries element for round charts. The property is used to provide names for the corresponding legend entries instead of the label property.

The step

- Replaces an empty value of the categoryLabel property in round charts with the value of the label property.
- Clears the value of label and blocks the use of this property for round charts.

Remove obsolete PCF properties

This step performs a series of transformations on PCF files.

numCols property

In release 9, the numCols property indicates an approximate number of letters that fit in the input.

The step:

- Replaces the property with width for Cell widgets derived from ValueCellTypeNode.
- Replaces the property with valueSectionWidth for Input widgets derived from ValueInputTypeNode.

If numCols is set to a value, it is translated to the em value, for example, 5 in numCols becomes 5em in width. If numCols is set to a Gosu expression, it is removed. Gosu expressions cannot be used for the width and valueSectionWidth properties.

checkSpelling property

The checkSpelling property became obsolete because most browsers now have a built-in spell check.

The step removes this property from text-related Input and Cell widgets.

flex property

The flex property, which was present in Cell and CellRef widgets, is no longer available.

The step removes the property and sets the new grow property to true if flex was greater than 0.

ReflectOnBottom property

The ReflectOnBottom property is no longer available in Toolbar widgets.

The step removes this property.

valueWidth property

The `valueWidth` property is no longer supported in `Input` widgets. You can configure the width with new width properties, like `valueSectionWidth`, `valueSectionMinWidth` and `valueSectionMaxWidth`.

The step removes the `valueWidth` property from the `Input` widget. After the property is removed, you must manually reconfigure the width using the new width properties.

equalWidth property

The `equalWidth` property is no longer supported in the `Layout` widget.

The step removes this property.

unique property

The `unique` property is no longer supported in the `PrivacyCell` widget.

The step removes this property.

Rename US English properties

The InsuranceSuite applications now contain all supported languages.

This step renames all US English properties files. For example, `display_en_US.properties` is renamed to `display.properties`.

Replace CurrentDate

This step replaces `GWBaseDateEnhancement.CurrentDate` with `gw.api.util.DateUtil.currentDate()` in Gosu files.

Schedule package upgrade

In InsuranceSuite 10, the package for `Schedule` has changed from `gw.api.productmodel.Schedule` to `gw.api.domain.Schedule`.

This step replaces references to `gw.api.productmodel.Schedule` with references to `gw.api.domain.Schedule` in PCF and Gosu files.

SimpleValuePopulator replace upgrade

This step replaces usages of `gw.webservice.pc.pc700.gxmodel.SimpleValuePopulator` with usages of `gw.webservice.pc.pc1000.gxmodel.SimpleValuePopulator` in Gosu files.

String Date replace upgrade

This step refactors custom castings from `String` to `Date` with the `Date.createDateTimeInstance` method in PCF and Gosu files.

Update default PCF properties

This step performs a series of transformations on PCF files.

numRows property

The default value for the `numRows` property in the `TextAreaInput` widget changed from `1` to `5`.

The step:

- Sets the numRows value explicitly to 1 if the default value was used.
- Does nothing, if the numRows value was specified.

clearEnabled property

The default value for the clearEnabled property changed from true to false.

The step sets the clearEnabled value explicitly to true if the default value was used.

Spaces between links

In release 9, adjacent links configured to look like HTML links are not separated by a space. Some applications use spaces or commas in link labels. In release 10, padding between all link elements is added unless the link style is set to noSpaceLink.

The step adds spaces for links where needed.

BillingCenter upgrade steps

These steps run only when you upgrade BillingCenter configuration.

Copy charge enhancement

This step copies ChargeEnhancement.gsx with the deprecated API from the resources to the upgraded customized configuration in the temporary directory.

Copy enhancements for single currency and multicurrency

Several Gosu exposed properties and methods changed because multicurrency plans were introduced.

The InsuranceSuite Upgrade Tools contain two upgrade steps, **Copy enhancements for single currency** and **Copy enhancements for multicurrency**, which copy the following enhancements for single currency and multicurrency from the resources to the upgraded customized configuration in the temporary directory:

- BillingPlanBuilderEnhancement.gsx
- BillingPlanEnhancement.gsx
- AgencyBillPlanBuilderEnhancement.gsx
- AgencyBillPlanEnhancement.gsx
- DelinquencyPlanBuilderEnhancement.gsx
- DelinquencyPlanEnhancement.gsx
- PaymentPlanBuilderEnhancement.gsx
- PaymentPlanEnhancement.gsx

The value of the MulticurrencyDisplayMode parameter in the config/config.xml file determines which step is run. If the parameter value is set to SINGLE, the **Copy enhancements for single currency** step copies the single currency version of the enhancements to the tmp directory. If the parameter value is set to MULTIPLE, the **Copy enhancements for multicurrency** step copies the multicurrency version of the enhancements to the tmp directory.

ClaimCenter upgrade steps

These steps run only when you upgrade ClaimCenter configuration.

Add missing ActivityRule categories

Typecodes require particular categories. These categories may not exist in the customized configuration if the customer modified typecodes that existed in the base configuration or added custom typecodes.

This step adds missing ActivityRule categories to the RuleContextDefinitionKey and TriggeringPointKey typelists.

PolicyCenter upgrade steps

These steps run only when you upgrade PolicyCenter configuration.

Add missing UWRule categories

Typecodes require particular categories. These categories may not exist in the customized configuration if the customer modified typecodes that existed in the base configuration or added custom typecodes.

This step adds missing `UWRule` categories to the `RuleContextDefinitionKey` and `TriggeringPointKey` typelists.

10.0.2 upgrade steps

These steps have `targetVersion` set to `10.0.2` and run when you upgrade from releases:

- 8.0
- 9.0
- 10.0.x

Remove Gosu coercion failures parameter

This step removes the `ThrowOnGosuCoercionFailuresForLongFloatDouble` configuration parameter from the `config/config.xml` file. This parameter was introduced in InsuranceSuite release 9.0.7.

Remove obsolete PCF properties 2

This step removes obsolete attributes from PCF files:

- `imeMode` from all elements
- `entireTreeAtClient` from `TreeView` elements
- `pageLinksRenderer` from `LocationGroup` elements

The `imeMode` attribute was used to set the `imeMode` CSS property on the corresponding HTML input. This CSS property has been deprecated and is no longer a supported web standard (it was mainly implemented in Internet Explorer and has never worked in Chrome). For more information, see https://developer.mozilla.org/en-US/docs/Web/CSS/ime-mode.

The `entireTreeAtClient` was used on `TreeView` elements to copy the entire tree to the client on initial render. This functionality was useful only when the expand/collapse state on the client and server could be different (because then the tree could be present on the client but collapsed). In InsuranceSuite 10, the client and server are kept in sync so this functionality is now obsolete.

The `pageLinksRenderer` was used on `LocationGroup` elements to alter the way third-level navigation (nested `LocationGroup` elements) rendered in InsuranceSuite 7. The names of the nested groups would appear as horizontal tabs by default and this attribute could be used to render them as a select control instead. The attribute does nothing in InsuranceSuite 8, 9 and 10 and is no longer needed. Now, third-level navigation is part of the west panel.

Targeted post on change

This step transforms PCF files as follows:

- It replaces the `target` attribute in `PostOnChange` elements with `deferUpdate="false"`.
- It removes the `refreshTarget` attribute from `AddButton`, `AddMenuButton`, `IteratorButtons` and `RemoveButton` elements.

General upgrade steps

These steps do not have `targetVersion` set and run when you upgrade from any release.

Copy annotation class for deprecated API

This step copies the `DeprecatedAndRestoredByUpgrade.gs` annotation class to `gsrc/gw/api/upgrade/enhancements` in the upgraded customized configuration in the temporary directory.

Reformat XML

After all the upgrade steps are executed, this step runs to clean up the structure of the XML configuration files, such as entities, typelists, and PCF files.

Rename display key

This step looks for display keys that changed between releases and renames them in the locale properties, PCF files and Gosu files.

Rename files differ case sensitivity

This step finds files with the same name but written in a different case and changes the name in the base and customized configurations to be exactly the same as the name in the new base configuration. Without this transformation Merge Tracker would mark the file in the upgraded base configuration as **GW_DELETE** and the file in the new base configuration as **GW_ADD**.

Script parameters enhancement

After you merge script parameters added by the customer in the `ScriptParameters.xml` file, you need to create a getter for each parameter.

This step checks `ScriptParameterPack` elements in `config/resources/ScriptParameters.xml` in the customized configuration. For elements added by the customer, the step creates an `Ex_ScriptParametersEnhancement.gsx` enhancement with the `get` property in `gsrc/gw/scriptparameter` in the upgraded customized configuration in the temporary directory.

The step returns an error when a type transformation is not supported, the content cannot be converted, or `config/resources/ScriptParameters.xml` does not exist. The step logs information when it finds a duplication while transforming types.

chapter 9

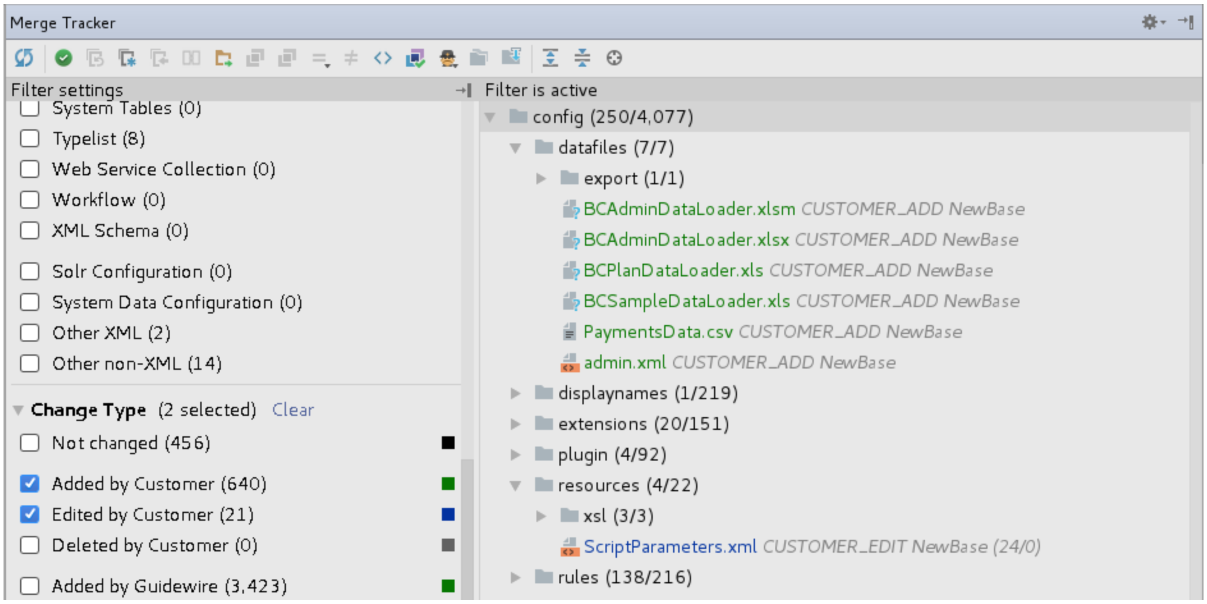
Merge Tracker tool

The Merge Tracker tool is a Guidewire Studio plugin that tracks progress and assists you in completing the manual part of the configuration upgrade process. Merge Tracker gives you a central location to view and work with the configuration files:

- Shows a complete list of all configuration files, along with the status of each file.
- Provides filters for viewing a portion of the configuration.
- Provides a central location to launch tools, such as Smart Merge, Smart Diff, and upgrade inspections.

Configuration file tree

The main panel of Merge Tracker displays the configuration file tree that matches filters selected in the filter pane. Files are shown once, regardless of the number of configurations in which they exist.



Merge Tracker

Filter settings

□ System Tables (0)
□ Typelist (8)
□ Web Service Collection (0)
□ Workflow (0)
□ XML Schema (0)

□ Solr Configuration (0)
□ System Data Configuration (0)
□ Other XML (2)
□ Other non-XML (14)

Change Type (2 selected) Clear

□ Not changed (456) ■

☑ Added by Customer (640) 🟩
☑ Edited by Customer (21) 🟦
□ Deleted by Customer (0) ■

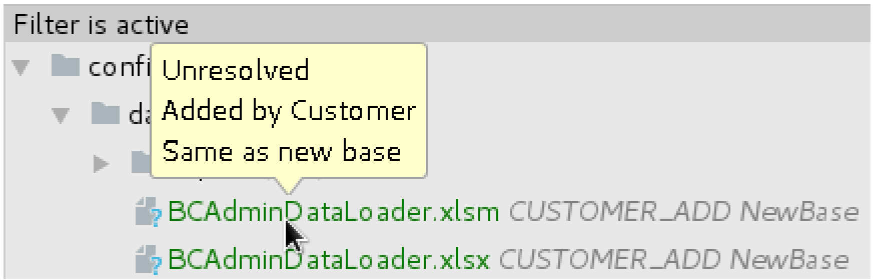
□ Added by Guidewire (3,423) 🟩

Filter is active

```
▼ config (250/4,077)
    ▼ datafiles (7/7)
        ▶ export (1/1)
          BCAdminDataLoader.xlsm CUSTOMER_ADD NewBase
          BCAdminDataLoader.xlsx CUSTOMER_ADD NewBase
          BCPlanDataLoader.xls CUSTOMER_ADD NewBase
          BCSampleDataLoader.xls CUSTOMER_ADD NewBase
          PaymentsData.csv CUSTOMER_ADD NewBase
          admin.xml CUSTOMER_ADD NewBase
    ▶ displaynames (1/219)
    ▶ extensions (20/151)
    ▶ plugin (4/92)
    ▼ resources (4/22)
        ▶ xsl (3/3)
          ScriptParameters.xml CUSTOMER_EDIT NewBase (24/0)
    ▶ rules (138/216)
```

Selecting the top-level directory selects all files that match the filters you set. Selecting a directory selects all files within that directory and its subdirectories.

Hovering the cursor over a file displays its basic details, like the resolve status and the current version of the file content.



Filter is active

confi

d

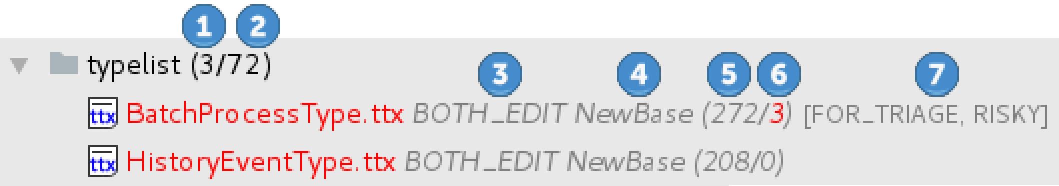
Unresolved
Added by Customer
Same as new base

BCAdminDataLoader.xlsm CUSTOMER_ADD NewBase

BCAdminDataLoader.xlsx CUSTOMER_ADD NewBase

If you want to learn more about a file, you can export its details to a CSV file. For more information, see “Contents of an exported file” on page 73.

Some details are also shown directly in the file tree.



typelist (3/72)

BatchProcessType.ttx BOTH_EDIT NewBase (272/3) [FOR_TRIAGE, RISKY]

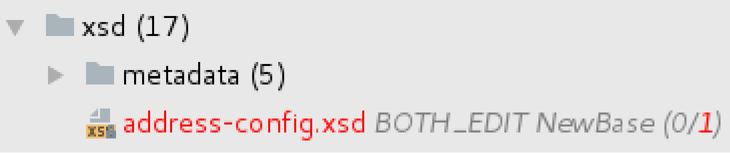
HistoryEventType.ttx BOTH_EDIT NewBase (208/0)

1. Number of files in the directory that match filters selected in the filter pane
2. Total number of files in the directory
3. Change type
4. Current version of the file content
5. Number of non-conflicting changes in the file.

   For BOTH_EDIT files, it is the sum of changes between the upgraded base version of the file and the upgraded customized version of the file (shown as **Customer changes** in Smart Merge) and changes between the upgraded base version of the file and the new base version of the file (shown as **New base changes** in Smart Merge).

   For BOTH_ADD files, it is the number of all non-conflicting and conflicting changes between the upgraded customized version of the file and the new base version of the file (shown as **All changes** in Smart Merge).
6. Number of conflicting changes in the file that cannot be merged and resolved automatically by Merge Tracker. For BOTH_ADD files, Merge Tracker does not display this counter.
7. Tags that are currently assigned to the file (in alphabetical order). Only first five tags are shown.

A padlock on a file's icon indicates that the file is locked (modifications to this file are not supported). A locked file that contains customer modifications is highlighted in red (the same way merge conflicts are highlighted).



xsd (17)

metadata (5)

address-config.xsd BOTH_EDIT NewBase (0/1)

Launch Merge Tracker in Guidewire Studio

Procedure

1. Run the following command in the top-level directory of the InsuranceSuite application where the InsuranceSuite Configuration Upgrade Tools are installed.

   ```
   gwb.bat studio
   ```
2. Open Merge Tracker. It will normally appear on the right side of the Studio window. If it does not appear, click the icon for showing the toolbar □ in the bottom left corner of the Studio window.

Next steps

For more information on how to use Merge Tracker, see “Merge Tracker tool” on page 57.

Merge Tracker command line options

To start the Merge Tracker tool from the command line, run `merge-tracker` from the `upgrade/bin` directory.

The following table lists Merge Tracker options.

Option	Action
`--merge-actions` *<ACTIONS>*	List of IDs of automatic merge and resolve actions to run (requires `-c auto-merge`). If not provided, all actions enabled by default are run.
`-X`	Print help including advanced and internal options (default: false)
`-c (--command) [report | list-merge-actions | auto-merge]`	Merge Tracker commands: • `report` - print details of the configuration files to the console (requires `-t`) • `list-merge-actions` - show available automatic merge and resolve actions • `auto-merge` - run all automatic merge and resolve actions enabled by default (requires `-t`). You can run specific or non-default actions by using the `--merge-actions` option.
`-h (--help)`	Print help (default: true)
`-t (--target)` *<PATH>*	Path to the target environment.

Merge Tracker toolbar

The Merge Tracker toolbar contains the following controls.

> **Note:** The toolbar controls are also available from the context menu.

Show filter settings

Opens the Merge Tracker filter pane that allows you to control which files and directories are shown in Merge Tracker. For details, see “Merge Tracker filters” on page 61.

Refresh

Refreshes the merge status of the file.

Resolve/Unresolve

Marks the file as resolved or unresolved. You typically mark a file as resolved after you merge changes and fix conflicts. This option only changes the resolved status of the file - it does not copy the file to the new customized configuration. The purpose of the resolved status is to have a general idea of the progress you are making in the upgrade.

Unresolve and revert to new base

Marks the file as unresolved and restores the file content to the new base version.

Copy original customized file

Copies the file from the upgraded customized configuration (in the temporary directory) to the new customized configuration. This button is enabled if there is a customized version of the file.

Revert to new base

Copies the file from the new base configuration to the new customized configuration.

Compare versions

Opens a window where you can choose two versions of the file to compare and then run a comparison tool for the chosen versions.

Export details

Exports the file details, like the merge status and the current version of the file content, to a CSV file. Instead of exporting details, you can quickly view basic information about a file by hovering the cursor over the file in the Merge Tracker tree. For more information on exported files, see “Contents of an exported file” on page 73.

Merge file

Runs a merge tool - Smart Merge, default Studio tool or an external tool. Which tool is launched depends on the file type and the Guidewire Studio configuration. For more details on file types supported by Smart Merge and the Merge Tracker configuration in Guidewire Studio, see “Smart Merge supported file types” on page 78and “Configuring Merge Tracker” on page 74 respectively.

Merge file and retire typecodes

Runs Smart Merge tool and automatically sets the `retired` attribute to `true` for all typecodes added in the new base configuration. This control is available only for typelist definition files (`.tti`, `.tix`, `.ttx`).

Mark file as renamed

Sets a name for the file and marks it as renamed. This control is available only if you select a renamed file from the customized configuration and the corresponding file with the default name from the new base configuration. After you mark the file as renamed, only the file with chosen name is shown in the Merge Tracker tree.

Unmark file as renamed

Reverts the action performed by the **Mark file as renamed** control. After you unmark the file as renamed, the renamed file from the upgraded customized configuration and the corresponding file with the default name from the new base configuration are shown again in the Merge Tracker tree.

Edit tags

Opens a window where you can add and edit *tags* (text strings separated by spaces) for the file. After applying tags, you can show or hide tagged files using the **Tags** filter in the Merge Tracker filter pane.

Merge and resolve files automatically

Opens a window where you can choose options for the merge and resolve process and then run it. Before using this option, select the files that you want to merge and resolve in the Merge Tracker file tree. If any filters are enabled in the filter pane, only files that match these filters are selected. The automatic merge and resolve process skips resolved files. To include a resolved file in the process, unresolve it.

Upgrade inspections

Opens a menu where you can run upgrade inspections and apply automatic upgrade inspection fixes. For more details on these actions, see “Use upgrade inspections” on page 83.

Copy path

Copies the absolute file path.

Get all versions

Creates a copy of the file for each configuration (upgraded base, new base, upgraded customized, new customized). This control can be useful if you need to share all the file versions with someone. For more information on the configurations, see “How the InsuranceSuite Configuration Upgrade Tools work” on page 11.

Expand all

Expands all nodes in the Merge Tracker tree.

Collapse all

Collapses all nodes in the Merge Tracker tree.

Scroll from source

Shows the file that is currently opened in the Studio editor in the Merge Tracker tree.

Merge Tracker filters

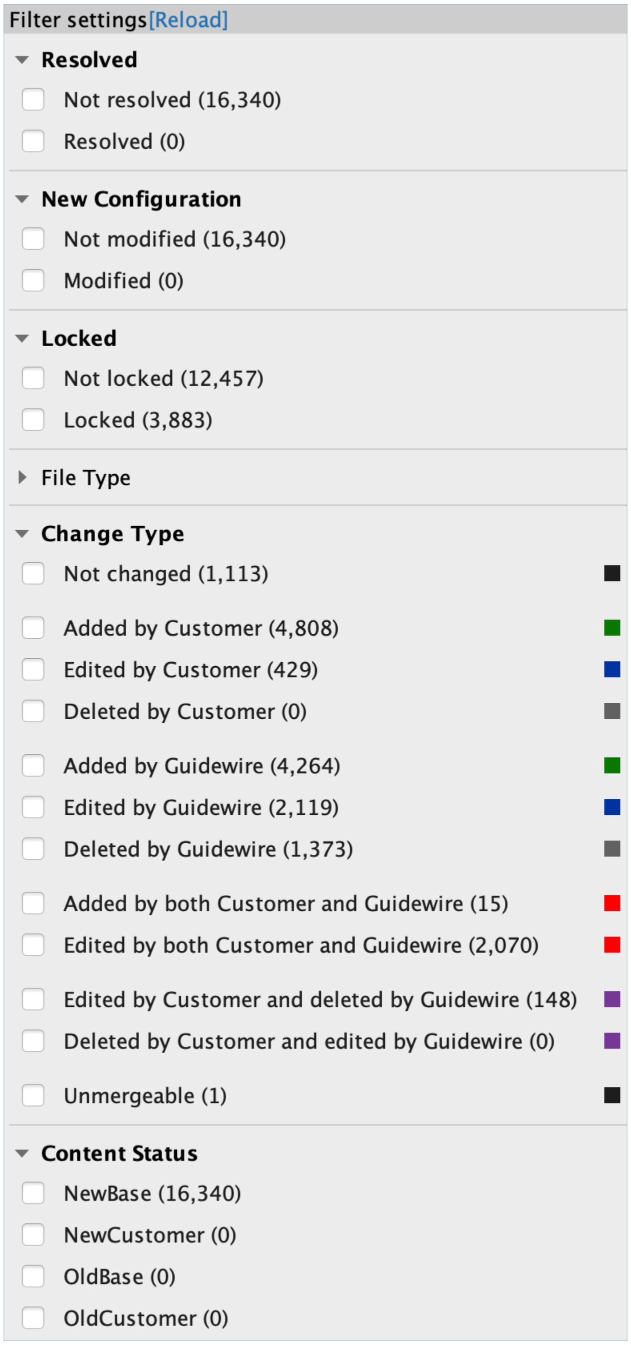
The Merge Tracker filter pane allows you to control which files and directories are shown in Merge Tracker.

During your upgrade project you may decide to organize your work into various categories or subprojects. For example, you may be focusing on upgrading one particular piece of functionality as a user story. In that case, you can use Merge Tracker filters to show only the portion of your configuration that is related to the user story.

Using the filters

To show the filter pane, select the filter button on the left-hand side of Merge Tracker.

Filters in the pane are divided into groups.



Filter settings [Reload]

▾ **Resolved**

□ Not resolved (16,340)

□ Resolved (0)

▾ **New Configuration**

□ Not modified (16,340)

□ Modified (0)

▾ **Locked**

□ Not locked (12,457)

□ Locked (3,883)

▸ File Type

▾ **Change Type**

□ Not changed (1,113) ⬛

□ Added by Customer (4,808) 🟩

□ Edited by Customer (429) 🟦

□ Deleted by Customer (0) ◼

□ Added by Guidewire (4,264) 🟩

□ Edited by Guidewire (2,119) 🟦

□ Deleted by Guidewire (1,373) ◼

□ Added by both Customer and Guidewire (15) 🟥

□ Edited by both Customer and Guidewire (2,070) 🟥

□ Edited by Customer and deleted by Guidewire (148) 🟪

□ Deleted by Customer and edited by Guidewire (0) 🟪

□ Unmergeable (1) ⬛

▾ **Content Status**

□ NewBase (16,340)

□ NewCustomer (0)

□ OldBase (0)

□ OldCustomer (0)

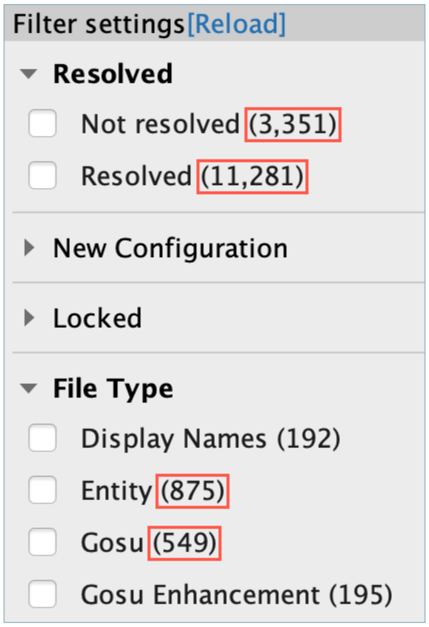
When you select more than one filter in a group, the files that meet at least one of the selected filtering criteria are shown (the OR rule). When you select filters in different groups, only the files that meet all the filtering criteria are shown (the AND rule).

Counters shown next to filters refresh dynamically based on the selected checkboxes in all the groups.

For more information on filter groups, see “Available Merge Tracker filters” on page 64.

Example: How the filters work

At the beginning, you do not have any filters selected. The counter for the **Resolved** filter shows 11,281, the counter for the **Not resolved** filter shows 3,351, the counter for the **Entity** filter shows 875 and the counter for the **Gosu** filter shows 549.



Filter settings [Reload]

Resolved

- □ Not resolved (3,351)
- □ Resolved (11,281)

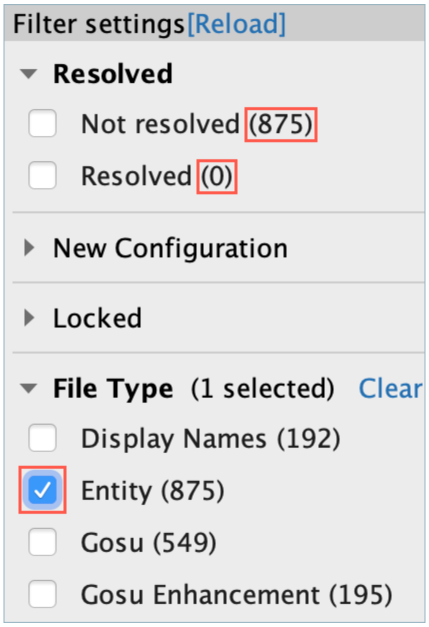
New Configuration

Locked

File Type

- □ Display Names (192)
- □ Entity (875)
- □ Gosu (549)
- □ Gosu Enhancement (195)

After you select the **Entity** filter, the counter for the **Resolved** filter changes to 0 and the counter for the **Not resolved** filter changes to 875 because all the 875 entities are not resolved.



Filter settings [Reload]

Resolved

- □ Not resolved (875)
- □ Resolved (0)

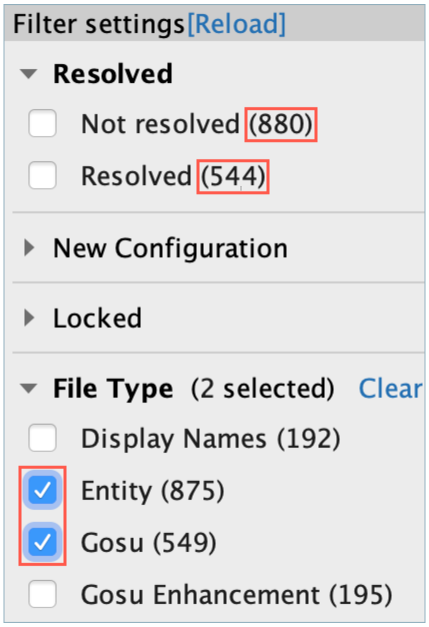
New Configuration

Locked

File Type (1 selected) Clear

- □ Display Names (192)
- ☑ Entity (875)
- □ Gosu (549)
- □ Gosu Enhancement (195)

After you also select the **Gosu** filter, the counter for the **Resolved** filter changes to 544 and the counter for the **Not resolved** filter changes to 880 because out of all the 549 Gosu files, 544 are resolved and 5 are still not resolved. So now the counter for the **Not resolved** filter takes into account both the entities and the Gosu files that are not resolved.



Filter settings[Reload]

- Resolved
  - □ Not resolved (880)
  - □ Resolved (544)
- New Configuration
- Locked
- File Type (2 selected) Clear
  - □ Display Names (192)
  - ☑ Entity (875)
  - ☑ Gosu (549)
  - □ Gosu Enhancement (195)

Available Merge Tracker filters

The Merge Tracker filter pane offers the following filter groups.

Resolved status

These filters allow you to quickly see which files have already been resolved. The purpose of the resolved status is to have a general idea of the progress you are making in the upgrade.

> **IMPORTANT** A resolved file is simply a file that you have marked resolved. The resolved status does not indicate whether a file has been merged or accepted.

New configuration

The **Modified** filter allows you to quickly see files that are different in the customized configuration and the new base configuration.

You can combine the new configuration filters with the resolved status filters. For example, you can select the **Modified** filter and the **Not resolved** filter to identify files that you need to review, merge and mark as resolved.

Locked filters

You can filter files that are marked locked in Guidewire Studio. Modifications to locked files are not supported. The filters make it easier to merge and resolve files without accidentally affecting modified locked files.

For example, you select all CUSTOMER_EDIT files and then want to move them to the target configuration and resolve. If some of the selected files are modified locked files, you get a warning. In this case, you can use the new filter to select and resolve only files that are not locked.

File type filters

You can filter files to be merged by file type, like entity or typelist.

Change type filters

This table describes the change type filters that are available in Merge Tracker. The Guidewire action column lists the change that Guidewire has made to the file since the prior version. The “Your action” column lists the change to the file in your implementation. For more information on configurations involved in the upgrade process, see “How the InsuranceSuite Configuration Upgrade Tools work” on page 11.

Change type **(Code shown in the Merge Tracker tree)**	**Guidewire action**	**Your action**	**Change description**
Not Changed **(NO_CHANGE)**	none	none	The file is the same in the base configuration, customized configuration and new base configuration. The new customized configuration already contains the file.
Added by Guidewire **(GW_ADD)**	add	none	The file exists in the new base configuration but it does not exist in the base configuration and customized configuration. The new customized configuration already contains the file.
Edited by Guidewire **(GW_EDIT)**	edit	none	The file in the new base configuration is different from the file in the base configuration. The new customized configuration already contains the new base version of the file. In file types supported by Smart Merge, formatting changes are ignored.
Deleted by Guidewire **(GW_DELETE)**	delete	none	The file exists in the base configuration and customized configuration but it does not exist in the new base configuration. The new customized configuration does not contain the file.
Added by Customer **(CUSTOMER_ADD)**	none	add	The file exists in the customized configuration but it does not exist in the base configuration and new base configuration.
Edited by Customer **(CUSTOMER_EDIT)**	none	edit	The file in the customized configuration is different from the file in the base configuration and new base configuration. In file types supported by Smart Merge, formatting changes are ignored.
Deleted by Customer **(CUSTOMER_DELETE)**	none	delete	The file exists in the base configuration and new base configuration but it does not exist in the customized configuration.
Added by both Customer and Guidewire **(BOTH_ADD)**	add	add	The file with a matching name and different content exists both in the customized configuration and new base configuration but the file does not exist in the base configuration (rare).
Edited by both Customer and Guidewire **(BOTH_EDIT)**	edit	edit	The file in the customized configuration and new base configuration is different from the file in the base configuration. In file types supported by Smart Merge, formatting changes are ignored.
Edited by Customer and deleted by Guidewire **(CUST_EDIT_GW_DELETE)**	delete	edit	The file does not exist in the new base configuration and the file in the customized configuration is different from the file in the base configuration.
Deleted by Customer and edited by Guidewire **(CUST_DELETE_GW_EDIT)**	edit	delete	The file does not exist in the customized configuration and the file in the new base configuration is different from the file in the base configuration. The new customized configuration already contains the new base version of the file.
Unmergeable **(UNMERGEABLE)**	change format of file	any	The file in the base configuration and customized configuration has a different format from the file in the new base configuration. The new customized configuration already contains the new base version of the file. Merge Tracker automatically marks certain files as unmergeable, such as `config.xml` and product model files in PolicyCenter. You can also specify a pattern in the Merge Tracker settings for files and directories that you want to mark as unmergeable. For details, see "Configuring Merge Tracker" on page 74.

Content status filters

You can filter files by their content. You can choose from the following filters.

NewBase

Content of the file in the new customized configuration is the same as the content of the file in the new base configuration.

NewCustomer

Content of the file in the new customized configuration is unique and differs from the content of the file in the upgraded base, upgraded customized and new base configurations. For example, the content is the result of merging customer changes from the upgraded customized configuration with Guidewire changes from the new base configuration.

OldBase

Content of the file in the new customized configuration is the same as the content of the file in the upgraded base configuration (in the temporary directory).

OldCustomer

Content of the file in the new customized configuration is the same as the content of the file in the upgraded customized configuration (in the temporary directory).

For example, the content status filters may be useful when you work on GW_DELETE files and use the **Copy original customized file** option to keep some of the files that Guidewire deleted. The content of the kept files changes from NewBase to OldBase. You can select the **OldBase** filter under the **Content Status** category to find the files you decided to keep and mark them as resolved.

Tag filters

You can filter files to be merged by *tags* (text strings separated by spaces). For more information on how to add tags to files, see “Merge Tracker toolbar” on page 59.

Custom filters

You can define your own filters for Merge Tracker. For more information, see “Manage custom filters” on page 66.

Manage custom filters

About this task

You can define custom filters for Merge Tracker.

Each filter has its own YAML file with configuration properties. Configuration files for custom filters are stored in `upgrade/etc/filters`. The name of the configuration file is used as the filter's label in the Merge Tracker filter pane.

A configuration file must contain the `pathPatterns` property. It is a list of patterns that are matched against paths of merge items. A pattern can be a standard file path or a regular expression supported by Java.

A template configuration file, `upgrade/etc/filters/sampleFilter.yaml.template`, can be used as the starting point for a new filter. In Merge Tracker, custom filters are listed in the **Custom Filters** group in the filter pane. After making changes to custom filters, you must refresh settings in Merge Tracker by selecting **Reload** at the top of the filter pane.

Add a custom filter

About this task

To add a new custom filter, you must create a YAML file with configuration properties.

Procedure

1. In Merge Tracker, go to **Custom Filters** in the filter pane and select **Open directory**.

2. In the directory, create a YAML file with a proper name. The name of the file is used as the filter's label in the Merge Tracker filter pane. If you prefer, you can duplicate the template file, `sampleFilter.yaml.template`, and then rename it.
3. Open the YAML file and add properties for your filter:
   - `pathPatterns` - a list of standard file paths and Java regular expressions for filtering files by path. Patterns are relative to the `modules/configuration` directory. Use only forward slashes (`/`) as path separators on all platforms.
   - `contentPatterns` - a list of Java regular expressions for filtering files by content.

   When you have `pathPatterns` and `contentPatterns` defined, the filter shows files that meet criteria from both properties (the AND rule). All patterns are case-sensitive.

   IMPORTANT The YAML file must have correct syntax. For example, use only spaces for indentation (tabs are not allowed). If the syntax is incorrect, Merge Tracker will not show the filter and will display a warning in the **Event Log**.

   For example

   ```
   pathPatterns:
     - config/web/pcf/.*
   contentPatterns:
     - width="[0-9]*%"
   ```
4. In Merge Tracker, select **Reload** at the top of the filter pane to display the new filter in **Custom Filters**.

Example

You can create a custom filter that shows files from a particular accelerator.

On Linux

1. Create a YAML file with patterns by running the following command in the root directory of the accelerator.

   ```
   (echo 'pathPatterns:' && find . -path "*/modules/configuration/*" -type f | sed -e 's/^.*\/modules\/
   configuration\//  - /') > MyAccelerator.yaml
   ```
2. In Merge Tracker, go to **Custom Filters** in the filter pane and select **Open directory**.
3. Copy `MyAccelerator.yaml` to the directory.
4. In Merge Tracker, select **Reload** at the top of the filter pane to display the **My Accelerator** filter in **Custom Filters**.

On Windows

1. Save a list of all accelerator files to a text file by running the following command in the root directory of the accelerator.

   ```
   dir "modules\configuration" /s /b /a-d /o:gn > accelerator_files.txt
   ```
2. In Merge Tracker, go to **Custom Filters** in the filter pane and select **Open directory**.
3. In the directory, create `MyAccelerator.yaml`.
4. Use a tool that supports the find and replace operation and make paths in the text file relative to the `modules/configuration` directory. Replace all path separators with forward slashes (`/`).
5. Open `MyAccelerator.yaml` and add the `pathPatterns` property with the relative paths. For example:

   ```
   pathPatterns:
     - config/extensions/entity/Extension1.etx
     - config/extensions/entity/Extension2.etx
   ...
   ```
6. In Merge Tracker, select **Reload** at the top of the filter pane to display the **My Accelerator** filter in **Custom Filters**.

Delete a custom filter

About this task

To delete a custom filter, you must delete the YAML file with the filter's configuration.

Procedure

1. In Merge Tracker, go to **Custom Filters** in the filter pane and select **Open directory**.
2. In the directory, delete the YAML file that has the same name as the filter you want to delete.
3. In Merge Tracker, select **Reload** at the top of the filter pane to remove the filter from **Custom Filters**.

Merging and resolving files

You can merge and resolve changes automatically or manually in Merge Tracker.

Note: The automatic merge and resolve process skips:

- Resolved files. To include a resolved file in the process, unresolve it.
- Files with the NewCustomer content status. For more information on content statuses, see “Content status filters” in “Available Merge Tracker filters” on page 64.

Merging and resolving files automatically from Merge Tracker

To run the automatic merge and resolve process from Merge Tracker:

1. In the Merge Tracker file tree, select the files that you want to merge and resolve. Selecting a directory selects all files within that directory and its subdirectories. If any filters are enabled in the filter pane, only files that match these filters are selected.
2. Click **(Merge and resolve files automatically)** in the Merge Tracker toolbar.
3. Select the actions you want to apply and continue.
4. Analyze the results to see how many files were processed.

Merging and resolving files automatically from the command prompt

To apply all automatic merge and resolve actions enabled by default:

1. Go to the `upgrade/bin` directory.
2. Run the following command.

   ```
   merge-tracker -c auto-merge -t <targetEnvironmentRootDirectory>
   ```

   For example.

   ```
   merge-tracker -c auto-merge -t pc_10
   ```

3. Analyze the results to see how many files were processed.

To run specific or non-default actions:

1. Go to the `upgrade/bin` directory.
2. To view available actions, run the following command.

   ```
   merge-tracker -c list-merge-actions
   ```

3. Run the selected actions with the following command.

   ```
   merge-tracker -c auto-merge -t <targetEnvironmentRootDirectory> --merge-actions <actionIds>
   ```

   For example.

   ```
   merge-tracker -c auto-merge -t pc_10 --merge-actions resolve-unchanged,delete-cd
   ```

4. Analyze the results to see how many files were processed.

> **IMPORTANT** Code generation fails when you start Guidewire Studio for the first time after you merge and resolve files automatically from the command prompt. To resolve this issue, run Guidewire Studio using the `gwb studio -x codegen` command.

For more information on Merge Tracker commands, see “Merge Tracker command line options” on page 59.

Merging and resolving files manually from Merge Tracker

Some changes cannot be merged and resolved automatically. You need to handle such changes manually in Merge Tracker.

For example, a file that was modified by both the customer and Guidewire can contain a conflicting change. In this case, you have two options:

1. Select the file and click **(Merge file)** in the Merge Tracker toolbar to run a merge tool and solve the conflict.
2. Select the file and click **(Copy original customized file)** in the Merge Tracker toolbar to copy the file from the upgraded customized configuration (in the temporary directory) to the new customized configuration.

For more information on the options available in Merge Tracker, see “Merge Tracker toolbar” on page 59. For more information on change types, see the “Change type filters” section in “Available Merge Tracker filters” on page 64.

Merging and resolving modified files from Merge Tracker

When you try to merge manually a file that has been modified, such as a file with the **changed-after-resolution** tag, Merge Tracker displays a window with the following options:

1. Reset all changes and start a new merge using the new base version of the file.
2. Keep changes and continue the merge using the new customized version of the file.

However, for CUSTOMER_ADD files, Merge Tracker does not display the window and it automatically continues the merge using the new customized version of the file. You do not get the number of conflicts in Smart Merge either. This is because Merge Tracker does not have the base and new base versions of the file that it can use for comparison.

If you want to reset all changes and start a new merge using the upgraded customized version of the file, use the **Copy original customized file** option in the Merge Tracker toolbar. For more information on the options available in Merge Tracker, see “Merge Tracker toolbar” on page 59.

You can include files with the **changed-after-resolution** tag in the automatic merge and resolve process by selecting the **Merge and resolve changed-after-resolution files for CUSTOMER_ADD, CUSTOMER_EDIT, BOTH_EDIT** action in the **Merge and resolve files automatically** window. When this action is enabled, Merge Tracker tries to merge conflicting and non-conflicting changes (for BOTH_EDIT changes) or copies the original customized file to the new customized configuration (for CUSTOMER_ADD and CUSTOMER_EDIT changes).

When you run the automatic merge and resolve process, modified files with the NewCustomer content status are skipped. For example, you want to upgrade an InsuranceSuite application from 9.0.x to 10.1.0. You modified a file in 9.0.x. The same file was also modified by Guidewire in 10.1.0 but the changes are non-conflicting. You run upgrade steps for the first time and then run the automatic merge and resolve process in Merge Tracker. The file does not have any conflicts so it is merged and resolved automatically and gets the NewCustomer content status because the content of the file is different from the content of the file in the upgraded customized and new base configurations. In other words, the content is the result of merging your changes with Guidewire changes. When you run the automatic merge and resolve process again, the file is skipped because it has the NewCustomer content status. If you make new changes to the file in 9.0.x and run the upgrade steps for the second time, the file is tagged as **changed-after-resolution** in Merge Tracker. Although the file changed, it still has the NewCustomer content status so by default it is skipped during the automatic merge and resolve process. You can include the file in the process by selecting the **Merge and resolve changed-after-resolution files for CUSTOMER_ADD, CUSTOMER_EDIT, BOTH_EDIT** action in the **Merge and resolve files automatically** window.

Locked files

Some configuration files are marked locked in Guidewire Studio. Modifications to locked files are not supported.

Merge Tracker does not prevent modifications to locked files but it offers some mechanisms that lower the risk of accidental changes:

- The automatic merge and resolve process skips over locked files and only marks them as resolved if they are unchanged.
- A confirmation window appears when you try to resolve a locked file that has modifications or to manually merge a locked file.
- The **Locked** filters make it easier to merge and resolve files without accidentally affecting modified locked files.

Merging and resolving different types of changes

When merging and resolving files you can handle changes in different ways depending on their type. Here are the steps that you can perform in Merge Tracker. Procedures that involve the **Merge and resolve files automatically** option can be also performed from the command prompt. For more information, see “Merging and resolving files” on page 68.

Added by Customer (CUSTOMER_ADD)

By default, the file does not exist in the new customized configuration.

Single action

To copy the added file from the upgraded customized configuration to the new customized configuration:

1. Select the file.
2. Click (**Copy original customized file**) and then ✓ (**Resolve**) in the toolbar.

Bulk action

To copy all CUSTOMER_ADD and CUSTOMER_EDIT files from the upgraded customized configuration to the new customized configuration:

1. Select all files in the file tree.
2. Click (**Merge and resolve files automatically**).
3. Select **Copy the customized version of the file and resolve the file for CUSTOMER_ADD and CUSTOMER_EDIT**.
4. Run the merge and resolve process.

Edited by Customer (CUSTOMER_EDIT)

By default, the new base version of the file already exists in the new customized configuration.

Single action

To copy the customized file from the upgraded customized configuration to the new customized configuration:

1. Select the file.
2. Click (**Copy original customized file**) and then ✓ (**Resolve**) in the toolbar.

Bulk action

To copy all CUSTOMER_ADD and CUSTOMER_EDIT files from the upgraded customized configuration to the new customized configuration:

1. Select all files in the file tree.
2. Click (**Merge and resolve files automatically**).
3. Select **Copy the customized version of the file and resolve the file for CUSTOMER_ADD and CUSTOMER_EDIT**.
4. Run the merge and resolve process.

Deleted by Customer (CUSTOMER_DELETE)

By default, the new base version of the file already exists in the new customized configuration.

Bulk action

To remove all CUSTOMER_DELETE files from the new customized configuration:

1. Select all files in the file tree.
2. Click (**Merge and resolve files automatically**).
3. Select **Delete the file for CUSTOMER_DELETE**.
4. Run the merge and resolve process.

Added by both Customer and Guidewire (BOTH_ADD)

By default, the new base version of the file already exists in the new customized configuration.

Single action

To merge the file from the customized configuration with the file from the new base configuration:

1. Select the file.
2. Click (**Merge file**) in the toolbar to run a merge tool.
3. Merge changes and save the result.

To copy the file from the upgraded customized configuration to the new customized configuration:

1. Select the file.
2. Click (**Copy original customized file**) and then (**Resolve**) in the toolbar.

Bulk action

To copy all BOTH_ADD files from the upgraded customized configuration to the new customized configuration:

1. Select all files in the file tree.
2. Click (**Merge and resolve files automatically**).
3. Select one of the options:
   - **Copy the customized version of the file and resolve the file for BOTH_ADD and BOTH_EDIT** if you want to copy all files added and edited by you
   - **Copy the customized version of the file if the same as the new base version and resolve the file for BOTH_ADD** if you want to copy all files added by you that are the same as the files added by Guidewire
4. Run the merge and resolve process.

Edited by both Customer and Guidewire (BOTH_EDIT)

By default, the new base version of the file already exists in the new customized configuration.

Single action

To merge the file from the customized configuration with the file from the new base configuration:

1. Select the file.
2. Click (**Merge file**) in the toolbar to run a merge tool.
3. Merge changes and save the result.

To copy the file from the upgraded customized configuration to the new customized configuration:

1. Select the file.
2. Click (**Copy original customized file**) and then (**Resolve**) in the toolbar.

Bulk action

To merge or copy all BOTH_EDIT files from the upgraded customized configuration to the new customized configuration:

1. Select all files in the file tree.
2. Click (**Merge and resolve files automatically**).
3. Select one of the options:
   - **Automatically merge changes if there are no conflicts and resolve the file for BOTH_EDIT** if you want to automatically merge all **BOTH_EDIT** files without conflicts
   - **Automatically merge changes if there are resolvable conflicts and resolve the file for BOTH_EDIT** if you want to automatically merge all **BOTH_EDIT** files that contain resolvable conflicts
   - **Copy the customized version of the file and resolve the file for BOTH_ADD and BOTH_EDIT** if you want to copy all files added and edited by you
4. Run the merge and resolve process.

Edited by Customer and deleted by Guidewire (CUST_EDIT_GW_DELETE)

There are two possible reasons why the file does not exist in the new base configuration:

1. Guidewire removed the file from the InsuranceSuite application.
2. Guidewire moved the file to a different folder.

If Guidewire has completely removed the file, review the *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 environment. Move your customized file to the same location in the working directory and make sure to match any case changes in the file name. When you refresh the merge status in Merge Tracker, 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.

Single action

To copy the customized file from the upgraded customized configuration to the new customized configuration:

1. Select the file.
2. Click (**Copy original customized file**) and then (**Resolve**) in the toolbar.

Merge progress tracking

Merge Tracker tracks merge progress by creating a separate `.status` file in the `modules/configuration/merge-statuses` directory for each configuration file. The structure under the `merge-statuses` directory corresponds to the directory structure of your configuration. This solution eliminates conflicts when multiple developers are involved in the merge process. Guidewire recommends that you manage the files under the `merge-statuses` directory with a version control system.

Status file structure

A status file contains the following columns with information about the configuration file for which it was created.

`new customer`

Path to the configuration file in the new customized configuration

`old base`

Path to the configuration file in the upgraded base configuration

`new base`

Path to the configuration file in the new base configuration

`resolution`

Shows if the configuration file is marked as resolved

`merge status`

Not used, maintained for backward compatibility

`content status`

Not used, maintained for backward compatibility

`changes`

Not used, maintained for backward compatibility

`conflicts`

Not used, maintained for backward compatibility

`renamed`

Not used, maintained for backward compatibility

`tags`

Names of tags that are assigned to the configuration file

`oldCustomerChecksum`

CRC32 checksum for the configuration file content in the upgraded customized configuration in the temporary directory

Export file details

You can export details of a configuration file or multiple configuration files to a CSV file from Merge Tracker or the command prompt.

Export file details from Merge Tracker

Procedure

1. In Merge Tracker, select the files for which you want to export details.
2. Click (**Export details**) in the Merge Tracker toolbar.

Export file details from the command prompt

Procedure

1. At the command prompt, go to the `upgrade/bin` directory.
2. To export details of all configuration files in the target environment, run the following command.

   ```
   merge-tracker -c report -t <targetEnvironmentRootDirectory> > <fileName>
   ```

   For example:

   ```
   merge-tracker -c report -t pc_10 > configFilesDetails.csv
   ```

Contents of an exported file

If you exported details from Merge Tracker, the file contains information about the configuration files that you selected for export. If you exported details from the command prompt, the file contains details of all configuration files.

An exported file has the following columns.

`MergeStatus`

Change type

`Resolved`
: Shows if the configuration file is marked as resolved

`ContentStatus`
: Current version of the file content. For more information on content statuses, see "Content status filters" in "Available Merge Tracker filters" on page 64.

`OldBase`
: Path to the configuration file in the upgraded base configuration

`OldCustomer`
: Path to the configuration file in the upgraded customized configuration

`NewBase`
: Path to the configuration file in the new base configuration

`NewCustomer`
: Path to the configuration file in the new customized configuration

`Changes`
: Total number of changes in the file. For BOTH_EDIT files, it is the total number of Guidewire and customer changes.

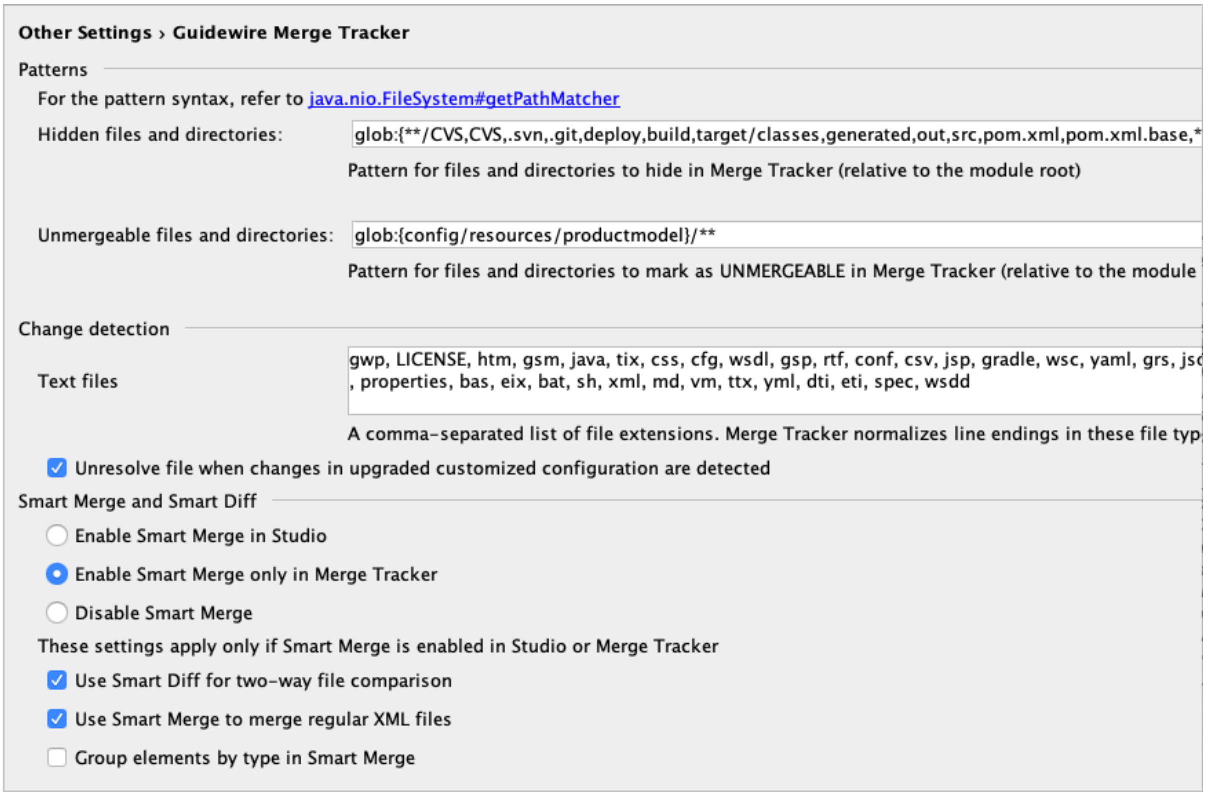
`Conflicts`
: Number of conflicting changes in the file that cannot be merged and resolved automatically by Merge Tracker

`Locked`
: Shows if the configuration file can be modified

`FileType`
: Shows the file type that Merge Tracker identified

Configuring Merge Tracker

You can change Merge Tracker settings in **File→Settings→Other Settings→Guidewire Merge Tracker**.



Other Settings › Guidewire Merge Tracker

Patterns

For the pattern syntax, refer to java.nio.FileSystem#getPathMatcher

Hidden files and directories: glob:{**/CVS,CVS,.svn,.git,deploy,build,target/classes,generated,out,src,pom.xml,pom.xml.base,*

Pattern for files and directories to hide in Merge Tracker (relative to the module root)

Unmergeable files and directories: glob:{config/resources/productmodel}/**

Pattern for files and directories to mark as UNMERGEABLE in Merge Tracker (relative to the module

Change detection

Text files: gwp, LICENSE, htm, gsm, java, tix, css, cfg, wsdl, gsp, rtf, conf, csv, jsp, gradle, wsc, yaml, grs, jso
, properties, bas, eix, bat, sh, xml, md, vm, ttx, yml, dti, eti, spec, wsdd

A comma-separated list of file extensions. Merge Tracker normalizes line endings in these file typ

☑ Unresolve file when changes in upgraded customized configuration are detected

Smart Merge and Smart Diff

○ Enable Smart Merge in Studio

● Enable Smart Merge only in Merge Tracker

○ Disable Smart Merge

These settings apply only if Smart Merge is enabled in Studio or Merge Tracker

☑ Use Smart Diff for two-way file comparison

☑ Use Smart Merge to merge regular XML files

☐ Group elements by type in Smart Merge

Patterns

Hidden files and directories, Unmergeable files and directories

Patterns for files and directories that you want to hide or mark as **UNMERGEABLE** in Merge Tracker. Patterns must follow the syntax supported by the `getPathMatcher` method of the `FileSystem` class. For more information on the syntax rules, see `java.nio.file.FileSystem#getPathMatcher(java.lang.String)` in the Java Platform documentation. The paths provided for the patterns are relative to the module root.

> **Note:** If you want to completely exclude files and directories from upgrade, you need to add proper patterns to the `upgrade.properties` file.

Change detection

Text files

A comma-separated list of file extensions. Merge Tracker normalizes line endings in these file types before calculating changes. This operation removes changes caused by differences in line endings among operating systems. The file types listed in this setting are opened automatically in Smart Merge when it is enabled.

Unresolve file when changes in upgraded customized configuration are detected

This option is useful when the customized configuration changes during the configuration upgrade process and you need to run upgrade steps again. When this option is enabled, resolved files that changed are unresolved automatically and tagged as **changed-after-resolution** in Merge Tracker. This option is enabled by default.

> **Note:** After you enable or disable this option, you must restart Guidewire Studio to apply the change.

Smart Merge and Smart Diff

Enable Smart Merge in Studio, Enable Smart Merge only in Merge Tracker, Disable Smart Merge

If Smart Merge is disabled, the merge tool configured in the Guidewire Studio settings is used.

You can use Smart Merge in the entire Guidewire Studio or only in Merge Tracker. If Smart Merge is enabled, it opens when you start merging a supported file type. For more information on supported file types, see “Smart Merge supported file types” on page 78.

Use Smart Diff for two-way file comparison, Use Smart Merge to merge regular XML files, Group elements by type in Smart Merge

These options apply only if Smart Merge is enabled in Studio or Merge Tracker.

You can enable Smart Merge for merging changes in regular XML files and Smart Diff for showing differences between two files.

If the grouping option is enabled, XML elements of the same type are by default shown one after another in the merge result pane in Smart Merge. Grouping is available only for Guidewire configuration files, like TTI files.

> **Note:** You can use the **Group elements by type** option in the bottom bar of the Smart Merge window to enable and disable grouping temporarily.

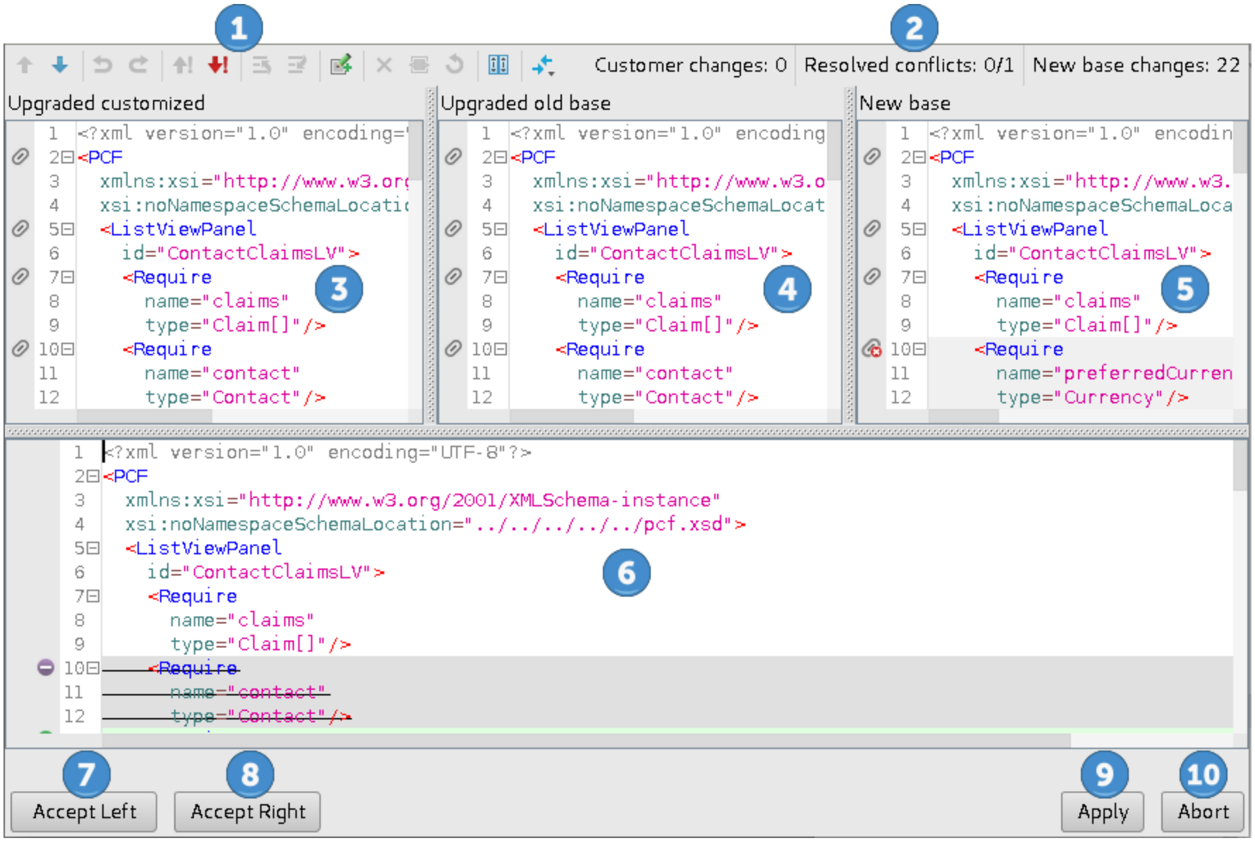
chapter 10

Smart Merge and Smart Diff tools

The Smart Merge tool is a three-way merge tool that understands the semantics of Guidewire configuration files, and uses that understanding to assist you in the merge process.

The Smart Diff tool is a comparison tool that understands the semantics of Guidewire configuration files.

Smart Merge user interface



Upgraded customized

```
<?xml version="1.0" encoding="
<PCF
  xmlns:xsi="http://www.w3.org
  xsi:noNamespaceSchemaLocatio
  <ListViewPanel
    id="ContactClaimsLV">
    <Require
      name="claims"
      type="Claim[]"/>
    <Require
      name="contact"
      type="Contact"/>
```

Upgraded old base

```
<?xml version="1.0" encoding
<PCF
  xmlns:xsi="http://www.w3.o
  xsi:noNamespaceSchemaLocat
  <ListViewPanel
    id="ContactClaimsLV">
    <Require
      name="claims"
      type="Claim[]"/>
    <Require
      name="contact"
      type="Contact"/>
```

New base

```
<?xml version="1.0" encodin
<PCF
  xmlns:xsi="http://www.w3.
  xsi:noNamespaceSchemaLoca
  <ListViewPanel
    id="ContactClaimsLV">
    <Require
      name="claims"
      type="Claim[]"/>
    <Require
      name="preferredCurren
      type="Currency"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<PCF
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../../../../../pcf.xsd">
  <ListViewPanel
    id="ContactClaimsLV">
    <Require
      name="claims"
      type="Claim[]"/>
~~    <Require~~
~~      name="contact"~~
~~      type="Contact"/>~~
```

1. Controls. For details, see “Smart Merge toolbar” on page 79.
2. Counters. For details, see “Smart Merge toolbar” on page 79.
3. File from the upgraded customized configuration in the temporary directory (left)

4. File from the upgraded base configuration in the temporary directory (base)
5. File from the new base configuration (right)
6. Merge result
7. Copies the file from the upgraded customized configuration to the new customized configuration and marks the file as resolved. This button performs the same operation as the **Copy original customized file** option in Merge Tracker. For more information on Merge Tracker controls, see "Merge Tracker toolbar" on page 59.
8. Copies the file from the new base configuration to the new customized configuration and marks the file as resolved.
9. Saves the merge result to the new customized configuration and marks the file as resolved
10. Closes Smart Merge without applying changes

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`)
- Generic XML (`.xml`)
- GX Model (`.xml`)
- System tables (`.xml`)
- Lookups (`.xml`)
- Schemas (`.xsd`)
- Web Service Collection (`.wsdl`)
- Gosu (`.gs`, `.gsp`)
- Gosu enhancements (`.gsx`)
- Gosu rules (`config/rules/*.*`)
- Gosu templates (`.gst`)
- YAML files (`.yaml`, `.yml`) - syntax highlighting only

Additionally, all file types listed in the **Text files** setting in the Merge Tracker settings are opened automatically in Smart Merge when it is enabled. For more information, see "Configuring Merge Tracker" on page 74.

Smart Diff command line options

To start the Smart Diff tool from the command line, run `smartdiff` from the `upgrade/bin` directory.

The `smartdiff` command line tool has two required arguments, *`left`* and *`right`*, corresponding to the two panes in the Smart Diff tool. The *`left`* parameter is the old version of the base configuration file. The *`right`* parameter is the new version of the base configuration file. Use `smartdiff -h` for help information about optional arguments.

The possible exit codes are listed in the following table.

Value	Meaning	Description
0	OK	Command ran successfully.

Value	Meaning	Description
3	UNKNOWN	The tool could not infer the merge type from the file names or file contents. When used with `--ignore-unrecognized`, this exit code allows you to write a script to use an alternative merge tool for unknown file types.
127	FAILED	General error.

Smart Merge command line options

To start the Smart Merge tool from the command line, run `smartmerge` from the `upgrade/bin` directory.

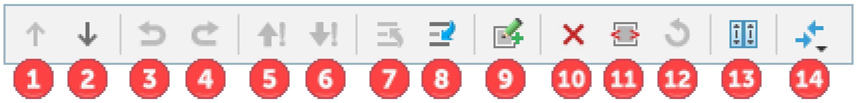
The `smartmerge` command line tool has three required arguments, *`base`*, *`left`* and *`right`*, corresponding to the three top panes in the Smart Merge tool. The *`base`* parameter is a base configuration file. The *`left`* parameter is the customer version of the same configuration file. The *`right`* parameter is the new version of the base configuration file. Use `smartmerge -h` for help information about optional arguments.

The possible exit codes are listed in the following table. You can create a wrapper script to run an alternative tool for the UNKNOWN exit code case.

Value	Meaning	Description
0	OK	Command ran successfully. This exit code also indicates that the merged file was saved with no unresolved conflicts.
1	INCOMPLETE	Indicates that the merged document was saved successfully, but some conflicts were not resolved.
2	CANCELLED	Indicates that the merged document was not saved before exit.
3	UNKNOWN	The tool could not infer the merge type from the file names or file contents. When used with `--ignore-unrecognized`, this exit code allows you to write a script to use an alternative merge tool for unknown file types.
127	FAILED	General error.

Smart Merge toolbar

Controls

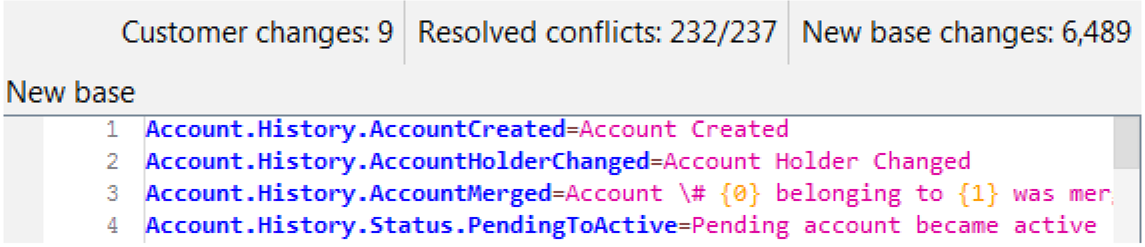


1. **Previous Difference**
2. **Next Difference**
3. **Undo**
4. **Redo**
5. **Previous Conflict**
6. **Next Conflict**
7. **Move Element Up**
8. **Move Element Down**
9. **Add Comment**
10. **Delete**
11. **Comment Out**
12. **Mark as Resolved**
13. **Synchronize Scrolling** - When this option is enabled and you click a linked node in one of the panes, other panes align automatically to show all the linked nodes side by side.

14. **Compare the Merge Result to One of the Source Documents** - You can use this option to compare the merge result with the upgraded base, upgraded customized or new base configuration without closing the Smart Merge window where you are currently working. The comparison result is shown in a new window.

Counters

For CUSTOMER_EDIT and BOTH_EDIT files, Smart Merge displays the following counters.



Customer changes: 9 | Resolved conflicts: 232/237 | New base changes: 6,489

New base

```
Account.History.AccountCreated=Account Created
Account.History.AccountHolderChanged=Account Holder Changed
Account.History.AccountMerged=Account \# {0} belonging to {1} was mer
Account.History.Status.PendingToActive=Pending account became active
```

Customer changes

Number of non-conflicting changes between the upgraded base version of the file and the upgraded customized version of the file

Resolved conflicts

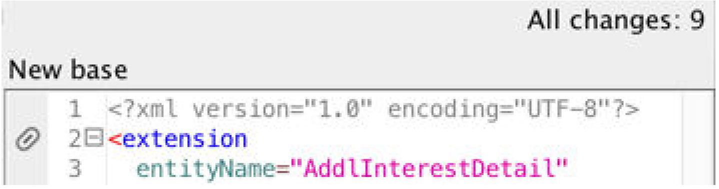
The first value is the number of conflicts that were resolved both automatically and manually. The second value is the number of conflicts between the upgraded customized version of the file and the new base version of the file. The number includes both conflicts that can be resolved automatically and conflicts that need to be resolved manually.

A difference between the upgraded customized version of the file and the new base version of the file is not enough to generate a conflict. A conflict occurs when the same part was modified in the upgraded customized version of the file and in the new base version of the file, and when this part is different in the upgraded base version of the file. For example, if a `name` attribute is set to `Property2` in the upgraded customized version of the file and to `Property1` in the upgraded base and new base versions of the file, it is a change. If a `name` attribute is set to `Property2` in the upgraded customized version of the file, to `Property1` in the upgraded base version of the file, and to `Property3` in the new base version of the file, it is a conflict.

New base changes

Number of non-conflicting changes between the upgraded base version of the file and the new base version of the file

For BOTH_ADD files, Smart Merge displays the following counters.



All changes: 9

New base

```
<?xml version="1.0" encoding="UTF-8"?>
<extension
  entityName="AddlInterestDetail"
```

All changes

Number of all non-conflicting and conflicting changes between the upgraded customized version of the file and the new base version of the file. The same number is shown for BOTH_ADD files in the Merge Tracker file tree.

Move an element in an XML file

About this task

When you merge an XML file in Smart Merge, you can move an entire element to a different position.

Procedure

1. In the pane showing the merge result, right-click the element you want to move and select **Move Element**.

   The element is marked with ✂.
2. To insert the element, click at a valid position (indicated by the horizontal line).

   Note: To abort the moving operation, choose **Cancel Move** from the right-click menu or press the `Esc` key.

Configuring Smart Merge and Smart Diff

For information on available Smart Merge settings, see “Configuring Merge Tracker” on page 74.

chapter 11

Upgrade inspections tool

Upgrade inspections are a set of Guidewire Studio inspections that identify issues after you complete merging and resolving changes. The inspections search the new customized configuration in the target environment for constructs that need to be upgraded. In Merge Tracker, you can run the upgrade inspections and then fix the issues manually or you can run the upgrade inspections with automatic fixes. From the command prompt, you can run code and upgrade inspections and save the results to a CSV file. You cannot apply fixes from the command prompt.

> **Note:** Do not start the InsuranceSuite application at this point. Guidewire Studio can run without connecting to the application server.

Use upgrade inspections

Run upgrade inspections and apply fixes manually from Merge Tracker

About this task

Use the **Run upgrade inspections** option from Merge Tracker in Guidewire Studio to find issues and, optionally, apply fixes.

Procedure

1. In the Merge Tracker file tree, select the files that you want to inspect. Selecting a directory selects all files within that directory and its subdirectories. If any filters are enabled in the filter pane, only files that match these filters are selected.
2. Click **(Upgrade inspections)→Run upgrade inspections** in the Merge Tracker toolbar.
3. Correct errors. You can defer fixing warnings.

Next steps

IMPORTANT Fixing initial errors and warnings may reveal more problems. Run the upgrade inspections repeatedly until no more errors are found.

Run upgrade inspections with automatic fixes from Merge Tracker

About this task

Run the **Apply automatic upgrade inspection fixes** option from Merge Tracker in Guidewire Studio to find issues and apply fixes automatically. This option allows you to automatically apply inspection fixes iteratively until all errors have been found and fixed.

Procedure

1. In the Merge Tracker file tree, select the files that you want to inspect and update. Selecting a directory selects all files within that directory and its subdirectories. If any filters are enabled in the filter pane, only files that match these filters are selected.
2. Click **(Upgrade inspections)→Apply automatic upgrade inspection fixes** in the Merge Tracker toolbar. All upgrade inspections run and inspection fixes are automatically applied. The upgrade inspections will run iteratively until all changes have been found and applied.
3. Verify that the **Apply automatic upgrade inspection fixes** process is complete by checking the Studio Event Log. When the process is complete, Merge Tracker will write a message to the Event Log indicating the number of fixes applied.

   For example:

   ```
   2:53:03 PM Automatic Upgrade Inspection Fixes applied 97 fixes in 1 pass in 6 minutes
   ```

Run inspections from the command prompt

About this task

Run code and upgrade inspections from the command prompt to find issues and save the results to a CSV file. You cannot apply fixes from the command prompt.

You have the following options:

- Run the default set of upgrade inspections
- Run a custom set of inspections by using an inspection profile
- Run inspections individually by providing their IDs

The command line tool saves the results to `inspectionResults/inspectionReport.csv`. If the `inspectionResults` directory exists, the tool deletes it before generating the CSV file.

Procedure

1. Go to the `upgrade/bin` directory.
2. If you want to run the default set of upgrade inspections, execute the following command.

   ```
   upgrade-inspections --source <InsuranceSuiteAppDir>
   ```

   For example (the `upgrade` directory is in the InsuranceSuite Configuration Upgrade Tools installation directory):

   ```
   upgrade-inspections --source ../../
   ```

3. If you want to run a custom set of inspections, execute the following command.

```
upgrade-inspections --source <InsuranceSuiteAppDir> --profile-path <inspectionProfileFile>
```

For example (the `upgrade` directory is in the InsuranceSuite Configuration Upgrade Tools installation directory):

```
upgrade-inspections --source ../../ --profile-path /ConfigUpgrade/CustomInspectionProfile.xml
```

4. If you want to run inspections individually, execute the following command.

```
upgrade-inspections --source <InsuranceSuiteAppDir> --inspection <inspectionId>
```

For example (the `upgrade` directory is in the InsuranceSuite Configuration Upgrade Tools installation directory)

```
upgrade-inspections --source ../../ --inspection AddMissingReifiedKeyword
```

Note: You can use the `--inspection` argument multiple times with different IDs to run a set of inspections. For IDs of upgrade inspections, see “Upgrade inspections reference” on page 85.

5. Analyze the results in `inspectionResults/inspectionReport.csv`.

Command line options for inspections

To start code and upgrade inspections from the command prompt, run `upgrade-inspections` in the `upgrade/bin` directory.

The following table lists options for inspections.

Option	Action
`-X`	Print help including advanced and internal options (default: false)
`-h (--help)`	Print help (default: true)
`-s (--source)` *<PATH>*	Relative path to the installation directory of an InsuranceSuite application (the directory where the `gwb` tool is located)
`-p (--profile-path)` *<PATH>*	Path to the file with the inspection profile. If not provided, only upgrade inspections are executed.
`-i (--inspection)` *<ID>*	ID of the inspection to run. Use this argument multiple times with different IDs to run a set of inspections. For IDs of upgrade inspections, see “Upgrade inspections reference” on page 85.

Upgrade inspections reference

Add missing reified keyword

Inspection ID: `AddMissingReifiedKeyword`

Gosu requires the `reified` keyword as a modifier on generic functions that are equivalent to parameterized Java functions that have type erasure. This inspection allows you to add the `reified` modifier automatically to functions that require it.

Convert Gosu interfaces to Gosu structures

Inspection ID: `EntityInterface`

This inspection finds Gosu interfaces used by `entity.implementsInterface` and converts them to Gosu structures because Gosu interfaces are not supported by `entity.implementsInterface`.

Find customer files copied to the new customized configuration and then removed from the tmp directory

Inspection ID: `DisplayRemovedCustomerFiles`

This inspection finds files that were added by the customer, copied into the new customized configuration and then removed from the `tmp` directory. It may be useful in a situation where the configuration upgrade is already in progress and you need to run the upgrade steps again because some files were removed, moved, or renamed in the customized configuration.

> **Note:** The inspection works only if the **Unresolve file when changes in upgraded customized configuration are detected** option is enabled in Merge Tracker settings before you run the upgrade steps again.

An example usage scenario:

1. You add a file, `CustomWidgets.pcf`, to the customized configuration.
2. You run upgrade steps. A temporary directory is created.
3. You launch Merge Tracker and then copy `CustomWidgets.pcf` to the new customized configuration.
4. You remove `CustomWidgets.pcf` from the customized configuration.
5. You run the upgrade steps again overwriting the temporary directory. The directory does not contain CustomWidgets.pcf any more but the file is still in the new customized configuration.
6. You go back to Merge Tracker and run the **Find files added and then removed by the customer** inspection to identify files that you need to remove from the new customized configuration.

Find modified locked files

Inspection ID: `LockedFiles`

Some configuration files are marked as locked by Guidewire. Modifications to locked files are not supported. This inspection finds locked files that have been modified, so they can be reviewed and corrected.

Fix final and static methods

Inspection ID: `RemoveFinalFromStaticGosuMethodDef`

Methods with `final` and `static` are invalid in Gosu. This inspection finds these methods and removes the `final` keyword.

Fix monetary expressions compared to zero

Inspection ID: `MonetaryAmountComparedToZero`

Types such as `MonetaryAmount` and `CurrencyAmount` have been added to Guidewire products to improve type safety of multicurrency features. Existing code that compares monetary values to unitless numbers must be reviewed and updated to take currency into account. This inspection fixes any cases where a monetary value is compared with a literal zero value, because the currency is irrelevant for the special case of comparing to zero. Comparisons with non-zero numeric values must be corrected manually.

Fix non-phone values without the GW or GWU prefix in the GWPhoneNumber context

Inspection ID: `PhoneToGwPhone`

This inspection finds non-phone values without the `GW` or `GWU` prefix that are used in the `GWPhoneNumber` context and allows you to:

- Add the missing prefix if the source or destination is an entity column of the `phone` type
- Wrap the value in the `gw.api.phone.upgrade.PhoneParser.exParsePhone` function if the source is a value of the `String` type

Fix renamed Gosu classes, methods, and field names

Inspection ID: `GenericApiDiffGosu`

This inspection finds and fixes renamed Gosu classes, methods, and field names. The inspection applies only to Gosu files. The complete list of renamed classes, methods, and field names updated by this inspection is specific to the version you are upgrading from and is based on the changes listed in the upgrade diff report. To obtain your custom Upgrade Diff Report, visit the Guidewire Community.

Fix the phone-type column without the Value suffix in the String context

Inspection ID: `PhoneToString`

This inspection finds columns of the `phone` type without the `Value` suffix that are used in the `String` context and allows you to add the missing suffix.

Inform about moved methods

Inspection ID: `InformAboutMovedMethods`

This inspection finds changed and moved methods and informs you where you need to move their customized logic. For example, the inspection finds the `IAccount.getShouldHoldAutomaticDisbursement(entity.Account)` method and informs you that you need to move the logic from this old method to the `AutomaticDisbursement.shouldHold` method.

Remove Gosu rule files not listed in order.txt

Inspection ID: `UnreachableRules`

Gosu rules that exist in the file system but do not appear in corresponding `order.txt` files are not visible in Guidewire Studio. This inspection finds and removes such rules.

Remove redundant conversion between String and TypeKey

Inspection ID: `StringTypekeyConversion`

This inspection finds and removes redundant conversions between `String` and `TypeKey` and replaces them with static `TypeKey` references.

Remove the ProxyClass property

Inspection ID: `ProxyClassProperty`

The Gosu `ProxyClass` property has been eliminated because Gosu classes can now be coerced directly to `java.lang.Class`. This inspection finds and removes usages of the `ProxyClass` property.

Replace Type properties used to access metatype properties and methods

Inspection ID: `EntityTypePropertiesAccess`

This inspection finds `Type` properties that are used to access entity and typelist metatype properties, and replaces them with `TYPE` references.

Replace XSD typeloader API

Inspection ID: `XsdTypeloaderApi`

This inspection finds usages of the old XSD typeloader API and replaces them with equivalent references to classes generated by the new XML code generator.

Replace calls to Gosu getter and setter methods with Gosu properties

Inspection ID: `ReplaceGetterAndSetterMethodsWithPropertySyntax`

This inspection finds and replaces calls to Gosu getter and setter methods with Gosu properties. For example, `gw.transaction.TransferFundsReversalWizardContext.setTransferFundTransaction(param)` is converted to `wizardContext.TransferFundTransaction = param`.

Replace obsolete date/time types

Inspection ID: `XsdDateTime`

Date/time types from the gw.xml.xsd.types package have been replaced by types from the gw.xml.date package. This inspection finds usages of the obsolete date/time types from the gw.xml.xsd.types package and replaces them with corresponding types from the gw.xml.date package.

Replace productmodel.XxxTypeValue types with generic types

Inspection ID: ProductModelTypeValue

This inspection finds usages of obsolete productmodel.XxxTypeValue types and replaces them with generic types.

Restore deleted display keys used in the previous release

Inspection ID: DeletedDisplayKeys

This inspection finds display key references in the Gosu code, such as DisplayKey.get("...")), that correspond to display keys deleted by Guidewire in the latest release. The inspection restores the display key values from the previous release.

Update PCF widgets missing type information

Inspection ID: MissingPcfTypes

This inspection finds input and cell elements in PCF files that are missing the valueType attribute. If the inspection can determine the data type stored by the element, it changes the element to a more strongly-typed version. For example, if a Cell element stores data of the String type, the inspection changes the element type to TextCell. The inspection also adds the valueType attribute if the element type does not match the data type. For example, if a Cell element stores data of the Integer type, the inspection changes the element type to TextCell and adds the valueType="java.lang.Integer". If the inspection cannot determine the data type, you need to change the element type manually. If the data type is dynamic.Dynamic, the inspection does not add the valueType attribute to the element.

Use denormalized policyPeriod.TaxAndSurchargesRPT

Inspection ID: UseTaxAndSurchargesRPT

This inspection finds usages of policyPeriod.allCosts.taxSurcharges.amountSum(policyPeriod.preferredSettlementCurrency) in PCF files and replaces them with the denormalized policyPeriod.TaxAndSurchargestRPT for better performance.

Use gw.pl.persistence.core.Bundle

Inspection ID: TransactionBundle

The gw.transaction.Bundle type has been replaced by gw.pl.persistence.core.Bundle. This inspection finds references to gw.transaction.Bundle and replaces them with references to gw.pl.persistence.core.Bundle.

chapter 12

Error log files

If an error occurs when running preupgrade, upgrade, Smart Merge, Smart Diff or Merge Tracker from a command line, the error is shown in the console and saved into a log file named `<toolName>-errors.log`. For example, all errors produced by preugprade steps are saved in the `preupgrade-errors.log` file.

You can also save warnings that occurred during preupgrade in the error log file by using the `--warnings-as-errors` option. For more information, see “Preupgrade command line options” on page 34.