Guidewire ClaimCenter®

ClaimCenter Globalization Guide

RELEASE 8.0.1

Guidewire®
Deliver insurance your way™

Product Name: Guidewire ClaimCenter
Product Release: 8.0.1
Document Name: *ClaimCenter Globalization Guide*
Document Revision: 10-February-2014

Contents

About ClaimCenter Documentation

The following table lists the documents in ClaimCenter documentation.

Document	Purpose
InsuranceSuite Guide	If you are new to Guidewire InsuranceSuite applications, read the *InsuranceSuite Guide* to understand the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with any Guidewire applications.
Application Guide	If you are new to ClaimCenter or want to understand a feature, read the *Application Guide*. Learn about features from a business perspective with links to other books as needed. The intended readers are everyone who works with ClaimCenter.
Upgrade Guide	Describes how to upgrade ClaimCenter from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing ClaimCenter application extensions and integrations.
New and Changed Guide	Describes new features and changes from prior ClaimCenter versions. Intended readers are business users and system administrators who want to understand new features and the upgrade impact for major releases. Consult the “Release Notes Archive” part of this document for changes in prior maintenance releases.
Installation Guide	Describes how to install ClaimCenter. The intended readers are everyone who installs the application for development or for production.
System Administration Guide	Describes how to manage a ClaimCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
Configuration Guide	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files. The intended audience is all IT staff and configuration engineers.
Globalization Guide	Describes how to configure ClaimCenter for a global environment. Learn about global locales, languages, date and number formats, names, currencies, addresses, and phone numbers. The intended readers are configuration engineers who work with locales and languages.
Rules Guide	Describes business rule methodology and the rule sets in ClaimCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
Contact Management Guide	Describes how each Guidewire InsuranceSuite application uses contacts for client data and vendor data. The intended readers are ClaimCenter implementation engineers and ContactManager administrators.
Best Practices Guide	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
Integration Guide	Describes the integration architecture, concepts, and procedures for integrating ClaimCenter with external systems and extending application behavior with custom programming code. The intended readers are system architects and the integration programmers who write web services code or plugin code in Gosu or Java.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Glossary	Industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.

Conventions in This Document

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

Support

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

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

part I

Introduction

chapter 1

Understanding Globalization

Globalization in ClaimCenter is the set of features and configuration procedures that make ClaimCenter suitable for operation in a global environment.

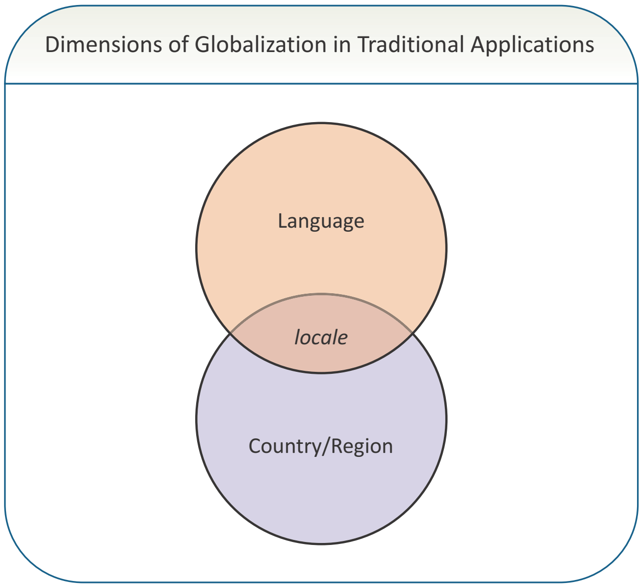
This topic includes:

Dimensions of Globalization

Traditionally, software solves the problem of operation in a global environment along two dimensions that intersect:

- **Language** – What alphabets and words to use for text in the user interface
- **Country/region** – How to format dates, times, numbers, and monetary values that users enter and retrieve

The intersection of these two dimensions – language with country/region – is known as *locale*.



Traditionally, applications let you select from a pre-configured set of locales. Java embodies this globalization dichotomy in the way that Java forms locale codes. A Java locale code combines an ISO 639-1 two-letter language code with an ISO 3166-1 two-letter country/region code.

For example, the locale code for U.S. English is `en-US`. A locale defines the language of text in the user interface as used in a specific country or region of the world. In addition, the locale defines the formats for dates, times, numbers, and monetary amounts as used in that same country/region.

Shortcomings of the Two Traditional Globalization Dimensions

In traditional applications, the two dimensions of globalization do not cover the following issues for software that operates in a global environment:

- Linguistic searching and sorting
- Phone number formats
- Address formats

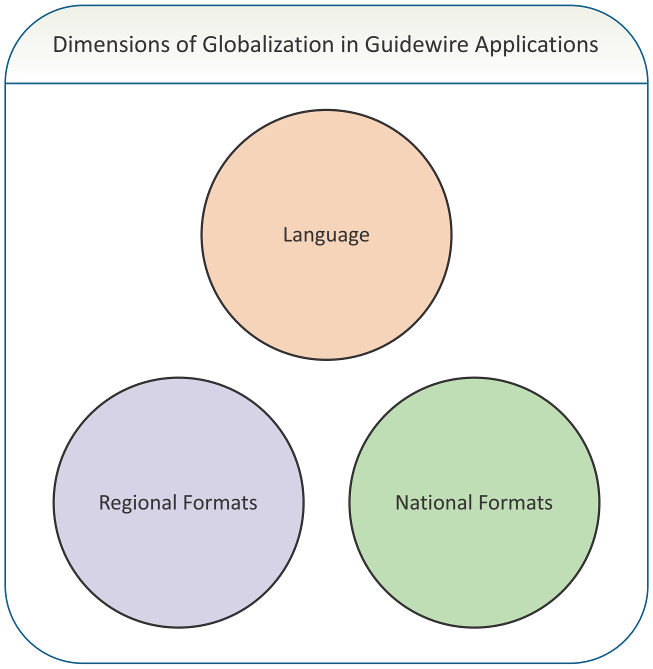
Furthermore, traditional applications allow users to select only pre-defined locales. For example, users typically cannot select French as the language, and, at the same time, select formats for dates, times, numbers, and monetary amounts used by convention in Great Britain.

Globalization Dimensions in Guidewire Applications

In contrast to traditional software, Guidewire applications solve the problem of operating in a global environment along three dimensions that are independent:

- **Language** – What alphabets and words to use for text in the user interface, as well as linguistic searching and sorting behavior.
- **Regional formats** – How to format dates, times, numbers, and monetary amounts that users enter and retrieve. Regional formats specify the visual layout of data that has no inherent association with specific countries/regions but for which formats vary by regional convention.

- **National formats** – How to format addresses and phone numbers. National formats specify the visual layout of data for which the country/region is inherent and the format remains the same regardless of local convention.



In Guidewire applications, you can select the language to see in ClaimCenter independently of the regional formats in which you enter and retrieve dates, times, numbers, and monetary amounts.

Phone numbers and addresses in ClaimCenter, however, use national (country) formatting, set through application configuration. For example, if you enter +86 in a phone field (China country code), then ClaimCenter shows the phone number using Chinese formatting. If you enter France for the country in an address field, then ClaimCenter shows address fields specific for France, including a CEDEX field.

User Selection of Language and Regional Formats

Within the Guidewire ClaimCenter application, each user can set the following:

- The language in which ClaimCenter displays labels and drop-down menu choices
- The regional formats that ClaimCenter uses to enter and display dates, times, numbers, monetary amounts, and names.

You set your personal preferences for display language and for regional formats by selecting **International** from the Options ⚙ menu at the top, right-hand side of the ClaimCenter screen. Then, you select options for language and regional formats independently. From the Options ⚙ menu, select **International**, and then select one of the following:

- **Language**
- **Regional Formats**

To take advantage of international settings in the application, you must configure ClaimCenter with more than one locale.

- ClaimCenter hides the **Language** submenu if only one language is installed.
- ClaimCenter hides the **Regional Formats** submenu if only one locale is configured.
- ClaimCenter hides the **International** menu option entirely if a single language is installed and ClaimCenter is configured for a single locale.

ClaimCenter indicates the current selections for **Language** and **Regional Formats** by placing a check mark to the left of the selected options and displaying them in gray.

Options for Language

In the base configuration, Guidewire has a single display language, English. To view another language in ClaimCenter, you must install a language module and configure ClaimCenter for that language. If your installation has more than one language, then you can select among them from the **Language** submenu. The `LanguageType` typelist defines the set of language choices that the menu displays.

If you do not select a display language from the **Language** submenu, then ClaimCenter uses the default language. The configuration parameter `DefaultApplicationLanguage` specifies the default language. In the base configuration, the default language is `English`.

Options for Regional Formats

If your installation contains more than one configured locale, then you can select a regional format for that locale from the **Regional Formats** submenu. At the time you configure a locale, you define regional formats for it.

Regional formats specify the visual layout of the following kinds of data:

- Date
- Time
- Number
- Monetary amounts
- Names of people and companies

The `LocaleType` typelist defines the names of regional formats that users can select from the **Regional Formats** submenu. The base configuration defines the following locale types:

- Australia (English)
- Canada (English)
- Canada (French)
- France (French)
- Germany (German)
- Great Britain (English)
- Japan (Japanese)
- United States (English)

Before a you select a regional format from the **Regional Formats** submenu, ClaimCenter uses the regional formats of the default locale. The configuration parameter `DefaultApplicationLocale` specifies the default locale. In the base configuration, the default locale is `United States (English)`. After you select your preference, you cannot revoke your choice and use the default locale again. Instead, you must select the default locale as your personal preference.

See also

- “Understanding Display Languages” on page 17
- “Working with Regional Formats” on page 57

Configuration Files Used for Globalization

ClaimCenter stores the various configuration files used for globalization in different locations in Studio. The following list describes the different globalization files and indicates their location in Studio (starting from **Project**).

Location in Studio	Configuration file	Description
configuration → config → Extensions → Typelist	`Languagetype.ttx`	List of defined languages
	`Currency.ttx`	Contains currency code and similar information for the supported currencies
	`PhoneCountryCode.ttx`	Phone country codes
	`State.ttx`	Address configuration
	`StateAbreviations.ttx`	
	`Jurisdiction.ttx`	
	`Country.ttx`	
configuration → config → Metadata → Typelist	`LocaleType.tti`	List of defined locales
configuration → config → Localizations	`localization.xml`	Currency formatting information for use with single currency rendering mode only. Studio contains multiple copies of this file, one for each locale,
	`display.properties`	Application display keys for a specific language
	`typelist.properties`	Application typecode names and descriptions for a specific language
	`*typelistName*.sort`	Sort order for typecodes in a typelist for a specific language
	`gosu.display.properties`	Gosu display keys used with error messages for a specific language
	`language.xml`	Collation configuration variant for a a specific language.
	`collations.xml`	Collation configuration for one or more languages, for one or more database types
configuration → config → currencies	`currency.xml`	Regional format overrides for monetary amounts in installations with multiple currencies
configuration → config → datatypes	`*dataType*.dti`	Data-type declarations for column types in the data model
configuration → config → fieldvalidators	`datatypes.xml`	Default values for `precision` and `scale` values in the default currency
	`fieldvalidators.xml`	Default field validator definitions and field validator definition overrides by country
configuration → config → geodata	`address-config.xml`	Address configuration
	`country.xml`	
	`zone-config.xml`	
	`*countryCode*-locations.txt`	

Location in Studio	Configuration file	Description
configuration → config → phone	`nanpa.properties`	Phone region
	`PhoneNumberMetaData.xml`	Phone region (Region 1 only)
configuration → config	`config.xml`	Configuration parameters related to globalization

Globalization Configuration Parameters in config.xml

Some configuration parameters in `config.xml` relate to general globalization features, such as display languages, regional formats, national formats, territorial data, and currencies. Other configuration parameters in `config.xml` relate to globalization features specific to ClaimCenter.

Configuration Parameters for General Globalization Features

The following parameters in `config.xml` relate to the general globalization features of ClaimCenter. For more information about these parameters, see "Globalization Parameters" on page 62 in the *Configuration Guide*

Parameter name	Sets...
`AlwaysShowPhoneWidgetRegionCode`	Whether the phone number widget in ClaimCenter displays a selector for phone region codes.
`DefaultApplicationCurrency`	Currency to use by default for new monetary amounts or whenever the use does not specify a currency for an amount.
`DefaultApplicationLangauge`	Language that the ClaimCenter shows by default for field labels and other string resources, unless the user selects a different personal preference for language.
`DefaultApplicationLocale`	Locale for regional formats in the application, unless the user selects a different personal preference for regional formats.
`DefaultCountryCode`	Country code to use for new addresses by default or if a user does not specify a country code for an address explicitly.
`DefaultNANPACountryCode`	Default country code for region 1 phone numbers.
`DefaultPhoneCountryCode` .	Country code to use for new phone numbers by default or if a user does not specify the country code during phone number entry.
`DefaultRoundingMode`	Default rounding mode for money calculations.
`MultiCurrencyDisplayMode`	Whether ClaimCenter displays a currency selector for money amounts. In single currency installations, there is no need for a currency selector as all amounts are in the default currency.

Configuration Parameters for Globalization Features Specific to ClaimCenter

The following configuration parameters in `config.xml` relate to globalization features unique to ClaimCenter and claims processing.

Parameter	See...
`EnableMulticurrencyReserving`	"EnableMulticurrencyReserving" on page 59 in the *Configuration Guide*
`PaymentRoundingMode`	"PaymentRoundingMode" on page 60 in the *Configuration Guide*
`ReserveRoundingMode`	"ReserveRoundingMode" on page 60 in the *Configuration Guide*

part III

Language Configuration

chapter 2

Working with Languages

ClaimCenter lets you configure support for multiple display languages. Display languages specify what alphabets and words to use for text in the user interface, as well as linguistic searching and sorting behavior. Generally, you configure ClaimCenter for display languages by installing language modules from Guidewire for those languages.

This topic includes:

Understanding Display Languages

The default display language in ClaimCenter is U.S. English. To display languages other than U.S. English, you must do the following:

- **Install additional display languages** – Install additional language modules from Guidewire for those languages by using the language module installer. Language modules contain localized screen and field labels and other string resources for a specific language.
- **Select the default display language for the application** – Set one of the languages as the default display language for ClaimCenter. Users see the application in the default language at the time they log in. Whenever they are logged in, users can select any installed language as their preferred display language.

Customer Support Limitations on Configuring ClaimCenter with Display Languages

Although ClaimCenter permits you to configure display languages of your choice manually, Guidewire does not provide customer support for any of the following:

- Configuring ClaimCenter with a language for which Guidewire does not provide a language module.

- Configuring ClaimCenter with a language for which Guidewire provides a language without installing the language module that Guidewire makes available to customers.

ClaimCenter has features that could allow you to manually configure display languages of your choice. However, those features exist to let you localize your own extensions to ClaimCenter for languages that you install from the language modules that Guidewire makes available.

This topic contains:

See also

- “Setting the Default Display Language for the First and Only Time” on page 23

Understanding Language Modules and Their Files

After you install a language module, Guidewire Studio displays the new module in the **Localizations** package of the project view. The module resides in a folder with the same name as the ZIP file for the language module. For example, if you install a language module named `cc-lang-zh-cn`, the installer creates the following folder for it:

```
config/
```

The types of files included in a language module can vary from module to module.

Language module folder	Language file
`Typelist`	• `Currency` • `LanguageType`
`currencies`	• `currency.xml`
Locale folder for that language	• `display.properties` • `gosu.display.properties` • `localization.xml` • `typelist.properties`

After you install the language module, the application `modules` directory contains the following:

- A copy of the language module ZIP file
- A language module folder with the same name as the language module ZIP file
- An archive file that contains copies of all the files that the language installation modified during the installation process
- A `base.zip` file

After you complete the language module installation, it is possible to delete the source language module ZIP file that you placed in the root application directory. However, do not delete any files from the `modules` directory that the language module placed there during the installation process. Most specifically, do not delete either of the language ZIP files (the source and archive file) or the `base.zip` file from the `modules` directory.

Understanding Language Hierarchies

You can configure ClaimCenter with language hierarchies in which one display language inherits localized values for display keys and other string resources from another language. Languages lower in a hierarchy contain only localized display keys and other string resources that differ from the localized values in the higher language. For example, consider Spanish as a *root language*, with *branch languages* for the variants of Spanish spoken in Spain and in Mexico.

Configuring The Root Language in a Language Hierarchy

In a language hierarchy, you configure the root language in the **Localizations** package with a localization folder that you name only with a lower-case, two-letter ISO language code. For example, the localization folder for the root English language would be:

```
configuration > config > Localizations > en
```

In the case of a Spanish language hierarchy, you start by creating an `es` folder in the **Localizations** package to configure the root Spanish language. In the `es` folder, provide values for all of the ClaimCenter display keys and other string resources localized to your version of the root Spanish language. The values that your provide might be equivalent to the values for Spanish spoken in Spain. Or, the values might be a form of international Spanish, suitable for native speakers throughout the Hispanic world.

Configuring Variant Branch Languages in a Language Hierarchy

In a language hierarchy, you configure the variant branch languages in the **Localizations** package with localization folders that you name with Java locale codes. A Java locale code specifies a language with a lower-case, two-letter ISO language code and a country with an upper-case two-letter ISO country code. For example, the localization folders for the variants of English spoken in Great Britain, Australia, and New Zealand would be:

```
configuration > config > Localizations > en_GB
configuration > config > Localizations > en_AU
configuration > config > Localizations > en_NZ
```

In the case of a Spanish language hierarchy, after you establish an `es` folder for the root Spanish language, create localization folders for variants of the Spanish language. Create an `es_ES` localization folder for Spanish spoken in Spain an `es_MX` localization folder for Mexican Spanish.

The `es_ES` localization folder most likely holds very few display keys or other string resources, because it inherits nearly everything from the `en` localization folder for root Spanish. In contrast, the `es_MX` localization folder most likely holds many more display keys and other resource strings than the `es_ES` localization folder. Compared to root Spanish, Mexican Spanish has many more linguistic variations than does the Spanish spoken in Spain.

Configuring a Root English Language for English-speaking Countries

Creating an English folder to support a locale for Britain is somewhat more complex as the base English locale is still `en_US` rather than `en`. In this case, Guidewire recommends that you do the following:

1. Create an `en` locale by copying the `en_US` properties files into that folder.
2. Create an `en_GB` folder, with the localized properties files that contain only the specific key/values for use in Great Britain.

Understanding the Language Module Installer

The language module installer is a command line tool that lets you install or upgrade installed display languages. The syntax for the command is:

```
gwcc install-localized-module -Dmodule.file=fileName.zip -Dinstall.type=installType
```

The command requires you to specify the following parameters.

Command Parameter	Description
`Dmodule.file`	The name of the ZIP file that contains the language module. Be certain to include the `.zip` extension. The file must be in the root of the ClaimCenter home directory.
`Dinstall.type`	The type of operation for the language module installer to perform: • `install` – Install a language module for a language that is not currently installed • `upgrade` – Upgrade an installed language with a newer version of the language module

See also

- “Installing Display Languages” on page 20
- “Upgrading Display Languages” on page 21
- “Activity Logging by the Language Module Installer” on page 24

Installing Display Languages

To view a display language in ClaimCenter other than U.S. English, you must install a language module from Guidewire for that language. To install a Guidewire language module, you must use the language module installer that Guidewire provides with ClaimCenter.

The language module installer performs the following actions whenever it installs a new language module:

- Verifies the integrity of the files in the language module before installing them
- Copies the files to their proper locations in the Guidewire home directory
- Modifies the files in place as necessary
- Verifies whether the language module was properly installed in your configuration of ClaimCenter.

After you install a language module, the installation is permanent. Language modules cannot be removed.

See also

- “Setting the Default Display Language for the First and Only Time” on page 23
- “Selecting a Personal Language Preference” on page 23
- “Upgrading Display Languages” on page 21

Installing a Language Module by Using the Language Module Installer

Use the language module installer to install a language module for a new language. You access the command to run the installer from the ***ClaimCenter***/bin directory.

To install a language module

1. Stop the ClaimCenter application server and Guidewire Studio, if either are running.
2. Copy the ZIP file that contains the language module to the root of the ClaimCenter home directory.
3. Open a command line and navigate to:
   ClaimCenter/bin
4. Enter the following command:
   ```
   gwcc install-localized-module -Dmodule.file=fileName.zip -Dinstall.type=install
   ```
   The command requires you to specify the following parameters.

Command parameter	Description
Dmodule.file	The name of the ZIP file that contains the language module. Be certain to include the .zip extension. The file must be in the root of the ClaimCenter home directory, regardless from where you run the language module installer.
Dinstall.type	The value must be install.

   The language module installer records all file operations that it executes in the command line window and in log files.
5. Restart the application server.

Result

ClaimCenter includes the language that you installed as a choice from the Options ⚙ menu.

See also

- “Setting the Default Display Language for the First and Only Time” on page 23
- “User Selection of Language and Regional Formats” on page 11

Installing Multiple Language Modules

It is possible to install multiple language modules to create a multi-lingual instance of ClaimCenter. Language modules for different display languages do not conflict with each other.

It is possible to reuse installed language modules across multiple countries. For example, for a hypothetical Swiss customer that requires French, German, and Italian language support, you need to install the following:

- French language module
- German language module
- Italian language module
- Switzerland country-support module

The German, French and Italian language modules do not contain any country-specific data. Thus, it is possible to reuse these modules for other German-, French-, or Italian-speaking customers. The Switzerland country support module contains configurations to enable basic country support for that country.

See also

- “Understanding Language Hierarchies” on page 18

Upgrading Display Languages

The steps to upgrade a display language to the latest release of an 8.0 language module depends on your starting release of ClaimCenter.

Starting release of ClaimCenter	Steps to upgrade
6.0.x	**1.** Upgrade your installation to 8.0.x. **2.** Install the 8.0.x language module in your 8.0 installation by using the 8.0 language module installer.
7.0.x	**1.** Uninstall the 7.0.x language module manually by following the instructions provided with the 8.0 documentation for ClaimCenter. **2.** Upgrade your installation to 8.0.x **3.** Install the 8.0.x language module in your 8.0 installation by using the 8.0 language module installer.
8.0.x	Upgrade your current 8.0 language module to the new 8.0.x language module by using the 8.0 language module installer.

This topic includes:

- “Upgrading a Language Module by Using the Language Module Installer” on page 22
- “Upgrade Properties File for the Language Module Installer” on page 22

See also

- To learn how to uninstall 7.0 language modules, see “Uninstalling Language Modules” on page 92 in the *Upgrade Guide*

Upgrading a Language Module by Using the Language Module Installer

Use the language module installer to upgrade a language module for a currently installed language. You access the command to run the installer from the *ClaimCenter*/bin directory.

Before you start the following procedure, be certain to set the properties in configuration file `upgrade.properties` according to our your needs.

To upgrade a language module

1. Stop the ClaimCenter application server and Guidewire Studio, if either are running.
2. Copy the ZIP file that contains the language module to the root of ClaimCenter home directory.
3. Open a command line and navigate to:

   ```
   ClaimCenter/bin
   ```
4. Enter the following command:

   ```
   gwcc install-localized-module -Dmodule.file=ZipFileName -Dinstall.type=upgrade
   ```

Command parameter	Description
`Dmodule.file`	The name of the ZIP file that contains the language module. Be certain to include the `.zip` extension. The file must be in the root of the ClaimCenter home directory, regardless from where you run the language module installer.
`Dinstall.type`	The value must be `upgrade`.

   The language module installer records all file operations that it executes in the command line window and in log files.
5. Restart the application server.

Result

The language module is upgraded. Be certain to review the log files that the language module install produces for upgrades that you might need to resolve.

See also

- "Upgrade Properties File for the Language Module Installer" on page 22

Upgrade Properties File for the Language Module Installer

The configuration file `upgrade.properties` has two properties that control how the upgrade language module installer performs upgrade operations:

- `upgrader.localized.resolve.conflicts`
- `upgrader.localized.staging.dir`

You must specify values for both parameters. The `upgrade.properties` file is in the following location:

```
ClaimCenter\modules\ant
```

Language Upgrade Property for Resolving Conflicts

Parameter `upgrader.localized.resolved.conflicts` in `upgrade.properties` specifies how the language module installer resolves conflicts encountered during an upgrade operation.

For example, you modified a display key that Guidewire modified in a newer version of a language module. Upgrade conflicts occur only for individual string resources with the same property name. String resources that you add to language configuration files and string resources that Guidewire adds in upgraded files do not cause upgrade conflicts, generally.

The `upgrader.localized.resolved.conflicts` parameter lets you specify the following alternatives for how the language module installer resolves upgrade conflicts:

- `customer` – Accept your modified language configuration file.
- `guidewire` – Accept the modified language configuration file from Guidewire.
- `manual` – Save both language configuration files for manual resolution by using a difference tool.

The following example configures the language module installer to always accept your versions of language configuration files if they contain upgrade conflicts.

```
upgrader.localized.resolve.conflicts=customer
```

Language Upgrade Property for the Language Upgrade Staging Directory

Parameter `upgrader.localized.staging.dir` in `upgrade.properties` lets you specify a directory for upgrade staging files that the language module installer creates. You must specify the complete path for the directory, for example:

```
upgrader.localized.staging.dir=C:\\tmp\upgrade-staging
```

You must specify a value for the `upgrader.localized.staging.dir` parameter, regardless what value you specify for the `upgrader.localized.resolved.conflicts` parameter. The language module installer always places copies of language configuration files in the staging directory. If upgrade conflicts occur, the content of the files in the directory vary depending on which resolution method the `upgrader.localized.resolved.conflicts` parameter specifies.

Resolution method	In the localization folder	In the staging directory
`customer`	Your current language file	New language file from Guidewire
`guidewire`	New language file from Guidewire	Your current language file
`manual`	Your current language file	New language file from Guidewire

If no update conflicts occur for a specific language configuration file, the language module installer copies the new language configuration file to the localization folder and to the staging directory.

Setting the Default Display Language for the First and Only Time

You must install the language that you want for your application default and set `DefaultApplicationLanguage` in `config.xml` before you start your 8.0 ClaimCenter server for the first time. You can install additional display languages later, but you cannot change the default application language.

The value that you set for `DefaultApplicationLanguage` must be a typecode in the `LanguageType` typelist. If you set the value of parameter `DefaultApplicationLanguage` to a value that does not exist as a `LanguageType` typecode, the application server refuses to start. Generally, when you install a language module, the installer adds a typecode for the installed language automatically to the `LanguageType` typelist automatically.

See also

- "Globalization Parameters" on page 62 in the *Configuration Guide*

Selecting a Personal Language Preference

Users of ClaimCenter can choose a preferred display language by selecting that language from the Options ⚙ menu. Language choices are available only for installed languages. A user's language preference overrides the default application language that you set system-wide with parameter

`DefaultApplicationLanguage` in `config.xml`. A user's choice for preferred display language persists between logging out and logging in again.

See also

- "Setting the Default Display Language for the First and Only Time" on page 23
- "User Selection of Language and Regional Formats" on page 11

Activity Logging by the Language Module Installer

Every time that the language module installer runs, it writes messages about its activities to the command console. When the installer finishes running, it writes the same messages to a log file.

This topic includes the following:

- "Activity Messages from the Language Module Installer" on page 24
- "Log Files that the Language Module Installer Creates" on page 26

Activity Messages from the Language Module Installer

Every time that the language module installer runs, it writes messages about its activities to the command console and to a log file. The activity messages have the following formats:

```
1:18:07,913 INFO INSTALL Localized Module
11:18:11,137 INFO Result:FINISHED
11:18:11,138 INFO
11:18:11,138 INFO Change Details:
11:18:11,138 INFO
11:18:11,138 INFO PREINSTALL STEP
11:18:11,138 INFO Validate Localized Module Checksum
11:18:11,138 INFO
11:18:11,138 INFO PREINSTALL STEP
11:18:11,138 INFO Validate Installation Environment and Version
11:18:11,139 INFO
11:18:11,139 INFO PREINSTALL STEP
11:18:11,139 INFO Validate Clean Environment without Previous Installation
11:18:11,139 INFO
11:18:11,139 INFO PREINSTALL STEP
11:18:11,139 INFO Ensure no typelist conflict
11:18:11,139 INFO
11:18:11,139 INFO PREINSTALL STEP
11:18:11,139 INFO Archive Current Module
11:18:11,139 INFO
11:18:11,140 INFO PREINSTALL STEP
11:18:11,140 INFO Archive Configuration Files
11:18:11,140 INFO ARCHIVE, source: C:/tmp/localized-module-test/.idea/modules.xml, target:
   px-lang-es_config_archive-8.0.0.dev.201306181134.zip
11:18:11,140 INFO ARCHIVE, source: configuration/pom.xml, target:
   px-lang-es_config_archive-8.0.0.dev.201306181134.zip
11:18:11,140 INFO ARCHIVE, source: C:/tmp/localized-module-test/pom.xml, target:
   px-lang-es_config_archive-8.0.0.dev.201306181134.zip
11:18:11,140 INFO
11:18:11,141 INFO INSTALL STEP
11:18:11,141 INFO Extract Module to the Application
11:18:11,141 INFO EXTRACT, source: C:/tmp/px-lang-es.zip/px-lang-es, target: px-lang-es
11:18:11,141 INFO
...
```

The first line of the log output always indicates the type of installer activity in progress, which, in this example, is `INSTALL`. Valid installer activity types are:

- `INSTALL`.
- `UPGRADE`

The second line of the log output indicates whether the installer was successful and whether the specified activity succeeded or failed. A value of `FINISH` indicates that the installer process completed successfully.

The following example further illustrates logging activity:

```
15:42:47,425 INFO UPGRADE STEP
15:42:47,426 INFO DisplayKey Merger
15:42:47,426 INFO DELETE_IN_MODULE, target: px-lang-es/config/locale/es_PE/gosu.display.properties
...
```

In the previous example log:

- Line 1 in the example log indicates the type of step, which, in this case, is `UPGRADE`.
- Line 2 of the example log lists the name of the current installer step.
- Additional lines list file operations that the installer performs on this `UPGRADE` step.

In the following example, the log shows there was an issue during the upgrade process due to a denial of file access. From viewing the log, you see that there is one file that was not accessible to the installer. As a consequence of the file access denial, the installer generates an error. The installer then restores any modified files to their original version before the upgrade started to preserve data integrity.

```
15:52:07,780 INFO UPGRADE Localized Module
15:52:17,434 INFO Result:FAILED
15:52:18,441 ERROR There was an error. The changes have been reverted
15:52:18,441 INFO
15:52:18,441 INFO Change Details:
15:52:18,441 INFO
15:52:18,441 INFO PREINSTALL STEP
15:52:18,441 INFO Validate Installation Environment
15:52:18,442 INFO
15:52:18,442 INFO PREINSTALL STEP
15:52:18,442 INFO Validate Base Module Present
15:52:18,442 INFO
15:52:18,442 INFO PREINSTALL STEP
15:52:18,442 INFO Validate Module Already Installed
15:52:18,442 INFO
15:52:18,442 INFO PREINSTALL STEP
15:52:18,442 INFO Archive Previous Module
15:52:18,442 INFO ARCHIVE, source: px-lang-es, target: px-lang-es-module-archive.zip
15:52:18,443 INFO
15:52:18,443 INFO PREINSTALL STEP
15:52:18,443 INFO Delete Staging Directory
15:52:18,443 INFO DELETE, target: C:/tmp/upgrade-staging
15:52:18,443 INFO
15:52:18,443 INFO UPGRADE STEP
15:52:18,443 INFO Set up the Staging Directory
15:52:18,443 INFO COPY_TO_STAGING, source: px-lang-es, target: C:/tmp/upgrade-staging
15:52:18,444 INFO
15:52:18,444 INFO UPGRADE STEP
15:52:18,444 INFO Localized Config File Upgrade
15:52:18,444 INFO COPY_TO_MODULE, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/locale
  /es_ES/newGuidewireSetting.properties, target: px-lang-es/config/locale/es_ES
  /newGuidewireSetting.properties
15:52:18,444 INFO COPY_TO_STAGING, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/locale
  /es_ES/newGuidewireSetting.properties, target: C:/tmp/upgrade-staging/config/locale/es_ES
  /newGuidewireSetting.properties
15:52:18,445 INFO COPY_TO_STAGING, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/extensions
  /typelist/Currency_es.ttx, target: C:/tmp/upgrade-staging/config/extensions/typelist
  /Currency_es.ttx
15:52:18,445 INFO COPY_TO_MODULE, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/locale
   /es_PE/newConfigFile.xml, target: px-lang-es/config/locale/es_PE/newConfigFile.xml
15:52:18,445 INFO COPY_TO_STAGING, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/locale
  /es_PE/newConfigFile.xml, target: C:/tmp/upgrade-staging/config/locale/es_PE/newConfigFile.xml
15:52:18,446 INFO COPY_TO_STAGING, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/locale
  /es_ES/display.properties, target: C:/tmp/upgrade-staging/config/locale/es_ES
  /display.properties
15:52:18,446 INFO COPY_TO_STAGING, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/locale
  /es_ES/localization.xml, target: C:/tmp/upgrade-staging/config/locale/es_ES/localization.xml
15:52:18,447 INFO DELETE_IN_STAGING, target: C:/tmp/upgrade-staging/config/locale/es_MX
  /localization.xml
15:52:18,447 INFO COPY_TO_STAGING, source: C:/tmp/px-lang-es-upgrade.zip/px-lang-es/config/extensions
  /typelist/LanguageType_es.ttx, target: C:/tmp/upgrade-staging/config/extensions/typelist
  /LanguageType_es.ttx
15:52:18,447 ERROR display.properties:
  C:\tmp\localized-module-upgrade-test\modules\px-lang-es\config\locale\es_ES\display.properties
  (Access is denied)
15:52:18,447 INFO
15:52:18,448 INFO REVERT STEP
15:52:18,448 INFO Delete Staging Directory
15:52:18,448 INFO DELETE, target: C:/tmp/upgrade-staging
15:52:18,448 INFO
```

```
15:52:18,448 INFO REVERT STEP
15:52:18,449 INFO Restore Archive Module
15:52:18,449 INFO RESTORE, source: px-lang-es-module-archive.zip, target: px-lang-es
15:52:18,449 INFO
```

Log Files that the Language Module Installer Creates

When it finishes running, the language module installer writes messages about its activities to a log file. It creates the log file in the standard application log directory, which by default is some variant of the following:

```
C:\tmp\gwlogs\ClaimCenter\logs
```

The log file has a file name similar to the following:

```
LocalizedModuleInstaller_20130308-15h52m05s.log
```

`_20130308-15h52m05s` is the date and time that the installer created the log file. This ensures that each log file is unique.

See also

- "Activity Messages from the Language Module Installer" on page 24

chapter 3

Localized Printing

Generating PDF documents in languages other than U.S. English typically requires additional configuration of your system and of Guidewire ClaimCenter.

This topic includes:

- "Printing Specialized Character Sets and Fonts" on page 27
- "Localized Printing in a Windows Environment" on page 28
- "Localized Printing in a Linux Environment" on page 30

Printing Specialized Character Sets and Fonts

PDF document generation in a specialized character set for languages other than U.S. English typically requires additional configuration. Adobe PDF (Portable Document Format) provides a set of fonts that are always available to all PDF viewers. This set of fonts includes the Courier, Helvetica, and Times font families and several symbolic type fonts. In some cases, however, it is possible that you need to download and install specialized font families to handle specific languages, Japanese, for example.

Guidewire does not provide fonts for use with Guidewire products. Any fonts that you use are part of the operating system platform as provided by a specific vendor. It is the operating system vendor that defines how you can use a specific font, and, under what circumstances. If you have questions about the acceptable use of a specific font, contact the operating system vendor that provided the font.

In particular:

- Guidewire assumes that appropriate fonts are made available for document printing and other features of the Guidewire applications.
- Guidewire expects that document fonts are provided and supported by the operating system vendor.
- Guidewire cannot and does not guarantee any of the fonts supplied as part of an operating system platform.

If you intend to print in a language not supported by one of the standard Adobe PDF fonts:

1. Install the required font.
2. Download and install the Apache Formatting Objects Processor (FOP).

Apache FOP is a print formatter of XML objects intended primarily for generating PDF output.

3. Configure Apache FOP and Guidewire ClaimCenter for the font that you installed.

See also

- For Japanese fonts, see:
  - `http://www.terra-intl.com/agel/2005/11/forrest_japanese.html`
  - `http://connie.slackware.com/~alien/slackbuilds/sazanami-fonts-ttf/pkg/12.0/`
- For additional information about Apache FOP and its font handling capabilities, see:
  - `http://xmlgraphics.apache.org/fop/trunk/fonts.html`
- For information on configuring Apache FOP and ClaimCenter for specific fonts, see:
  - "Localized Printing in a Windows Environment" on page 28
  - "Localized Printing in a Linux Environment" on page 30

Localized Printing in a Windows Environment

Suppose, for example, that you want to print PDF documents in Russian.The Russian language uses the Cyrillic character set. The default font for PDF generation does not support the Cyrillic character set. Therefore, you need to customize the Apache Formatting Objects Processor (FOP) application so that it uses fonts that do support the Cyrillic character set.

Fortunately, the generic Microsoft Windows Arial TrueType font family (normal, bold, italic, bold-italic) does support Cyrillic. If you work in a Windows environment, you can simply use the Arial TrueType font. To obtain a font that supports a particular language requirement but which is not currently installed as part of your operating system, contact your operating system vendor.

The following example illustrates how to configure Apache (FOP) and Guidewire ClaimCenter to print documents in Russian by using Cyrillic characters in a Windows environment.

- Step 1: Download and Install Apache FOP on Windows
- Step 2: Configure TTFReader
- Step 3: Generate FOP Font Metrics Files
- Step 4: Register the Fonts with Apache FOP
- Step 5: Register Your Apache FOP Configuration and Font Family with ClaimCenter
- Step 6: Test Your Configuration

This example assumes the following:

- Apache FOP exists on your machine.
- The `fop.jar` is on the class path.
- The Arial fonts exist in `C:\WINDOWS\Fonts`.
- The fonts are TrueType fonts.

The process for non-TTF FOP supported fonts is slightly different. See the Apache FOP documentation for more information.

The example also assumes the following:

- You have a working Guidewire ClaimCenter application.
- You have installed the proper language module for your Guidewire application.

Step 1: Download and Install Apache FOP on Windows

Download Apache FOP from the following Apache web site:

```
http://xmlgraphics.apache.org/fop/download.html
```

Guidewire currently supports Apache FOP 0.95. Download the binary version of FOP 0.95. Use the instructions provided by Apache to install the Formatting Objects Processor (FOP).

Step 2: Configure TTFReader

After you download and install Apache FOP, do the following:

1. Make a copy of `fop.bat` (in the root installation directory) and rename it `ttfreader.bat`.
2. Open `ttfreader.bat` and change the last line to read:

```
"%JAVACMD%" %JAVAOPTS% %LOGCHOICE% %LOGLEVEL% -cp "%LOCALCLASSPATH%"
        org.apache.fop.fonts.apps.TTFReader %FOP_CMD_LINE_ARGS%
```

   Essentially, you change `org.apache.fop.cli.Main` to `org.apache.fop.fonts.apps.TTFReader`. You perform this step so the code that generates font metrics works correctly.

Step 3: Generate FOP Font Metrics Files

FOP requires font metrics to use a font. You must give it the font metrics. To generate the font metrics, run the following commands. These commands activate the Apache FOP TTFReader.

```
ttfreader.bat -enc ansi C:\WINDOWS\Fonts\arial.ttf   C:\fopconfig\arial.xml
ttfreader.bat -enc ansi C:\WINDOWS\Fonts\ariali.ttf  C:\fopconfig\ariali.xml
ttfreader.bat -enc ansi C:\WINDOWS\Fonts\arialbi.ttf C:\fopconfig\arialbi.xml
ttfreader.bat -enc ansi C:\WINDOWS\Fonts\arialbd.ttf C:\fopconfig\arialbd.xml
```

This generates metrics files for the Arial font family and stores those metrics in the `C:\fopconfig` directory. Do not proceed until you see the following files in the `fopconfig` directory:

```
arial.xml
arialbd.xml
arialbi.xml
ariali.xml
```

Step 4: Register the Fonts with Apache FOP

You must register the fonts that you installed with Apache FOP. The following example registers the Arial font family with Apache FOP.

1. Create a configuration file for Apache FOP by copying the following file:

```
C:\fop_install\conf\fop.xconf
```

   into the following directory:

```
C:\fopconfig\
```

2. Open `fop.xconf` in an XML editor and find the `<fonts>` section. It looks similar to the following:

```
<fonts>
  ...
```

3. Enter the following font information in the appropriate place.

```
<fonts>
  <font metrics-url="c:\\fopconfig\\arial.xml" kerning="yes" embed-file="arial.ttf">
    <font-triplet name="Cyrillic" style="normal" weight="normal"/>
  </font>
  <font metrics-file="c:\\fopconfig\\arialbd.xml" kerning="yes" embed-file="arialbd.ttf">
    <font-triplet name="Cyrillic" style="normal" weight="bold"/>
  </font>
  <font metrics-url="c:\\fopconfig\\ariali.xml" kerning="yes" embed-file="ariali.ttf">
    <font-triplet name="Cyrillic" style="italic" weight="normal"/>
  </font>
  <font metrics-file="c:\\fopconfig\\arialbi.xml" kerning="yes" embed-file="arialbi.ttf">
    <font-triplet name="Cyrillic" style="italic" weight="bold"/>
  </font>
```

```
</fonts>
...
```

If you do not want to embed the font in the PDF document, then do not include the `embed-url` attribute.

4. Stop and start the application server so that these changes take effect.

Step 5: Register Your Apache FOP Configuration and Font Family with ClaimCenter

You must register your Apache FOP configuration file and font family with ClaimCenter. The following example registers the Cyrillic font family with ClaimCenter.

1. Open ClaimCenter Studio and search for `config.xml`.
2. Find the following parameters in `config.xml` and set them accordingly:

Parameter	Description	Example value
`PrintFontFamilyName`	The name of the font family for the custom fonts as defined in your FOP configuration file.	Cyrillic
`PrintFOPUserConfigFile`	The fully qualified path to a valid FOP configuration file.	C:\fopconfig\fop.xconf

Step 6: Test Your Configuration

After you perform the listed configuration steps, Guidewire recommends that you test that you are able to create and print a PDF that uses the correct font. To test the Apache FOP configuration, you must have a ClaimCenter implementation that supports the Russian locale.

Localized Printing in a Linux Environment

The following example illustrates how to configure Guidewire ClaimCenter and the Apache Formatting Objects Processor (FOP) to print Japanese characters in a Linux environment. This example includes the following steps:

- Step 1: Download and Install the Required Fonts
- Step 2: Download and Install Apache FOP on Linux
- Step 3: Configure the Font
- Step 4: Register the Font with Apache FOP
- Step 5: Register Your Apache FOP Configuration and Font Family with ClaimCenter
- Step 6: Test Your Configuration

The example also assumes the following:

- You have a working Guidewire ClaimCenter application.
- You have installed the proper language pack for your Guidewire application.

Before Starting

Guidewire recommends that you use a package manager to manage the download and installation of the necessary application files and packages on Linux. One such package manager is `yum`, which works with the following Linux distributions, among others:

- Fedora
- CentOS-5
- Red Hat Enterprise Linux 5 or higher

In any case, chose a package manager that works with your particular Linux distribution.

Step 1: Download and Install the Required Fonts

If it does not already exist in your operating system, you must obtain and install a font that supports the language in which you want to print.

1. Obtain a font from your operating system vendor that supports your particular language requirement.

   For example, you need to install a font that supports Japanese characters to print generated documents in Japanese. The following are examples of fonts that print Japanese characters:

   - IPA Gothic
   - Sanazami

2. If you use the `yum` package manager, substitute the actual font name for *`FONT-NAME`*:

   ```
   yum clean all
   yum install [FONT-NAME]
   ```

   If you use a package manager other than `yum`, use the install commands specific to your particular Linux distribution.

See also

- For Japanese fonts, see:
  - `http://www.terra-intl.com/agel/2005/11/forrest_japanese.html`
  - `http://connie.slackware.com/~alien/slackbuilds/sazanami-fonts-ttf/pkg/12.0/`

Step 2: Download and Install Apache FOP on Linux

To install Apache FOP in a Linux environment:

1. Download Apache FOP from the following Apache web site:

   ```
   http://xmlgraphics.apache.org/fop/download.html
   ```

   Guidewire currently supports Apache FOP 0.95. Download the binary version of FOP 0.95.

2. Unpack the ZIP file into the desired directory:

   ```
   mkdir /usr/local/fop-0.95
   cd /usr/local/fop-0.95
   cp /tmp/fop-0.95-bin.zip .
   unzip fop-0.95-bin.zip
   ```

3. Perform the following test to be certain that Apache FOP is working correctly:

   ```
   ./fop -fo examples/fo/basic/readme.fo -awt
   ```

Step 3: Configure the Font

Enter commands such as the following to generate the font configuration file. Use commands that are specific to your font. The following example is specific to the IPA Gothic font family.

```
cd /usr/local/fop-0.95
cp /usr/share/fonts/ipa-gothic/ipag.ttf .
java -cp build\fop.jar:lib\avalon-framework-4.2.0.jar:lib
        \commons-logging-1.0.4.jar:lib\xmlgraphics-commons-1.3.1.jar:lib
        \commons-io-1.3.1.jar org.apache.fop.fonts.apps.TTFReader -ttcname
        "IPA Gothic" ipag.ttf ipag.xml
```

Step 4: Register the Font with Apache FOP

You must register the font that you installed with Apache FOP. The following example registers the IPA Gothic font family with Apache FOP.

1. Open `fop.xconf` for editing. To use the `vi` editor, enter the following at a command prompt:

   ```
   vi conf/fop.xconf
   ```

2. Add the following lines to `fop-xconf` in the `<fonts>` section:

```
<font metrics-url="/usr/local/fop-0.95/ipag.xml" kerning="yes"
        embed-url="/usr/local/fop-0.95/ipag.ttf">
  <font-triplet name="IPAGothic" style="normal" weight="normal"/>
  <font-triplet name="IPAGothic" style="normal" weight="bold"/>
</font>
```

Step 5: Register Your Apache FOP Configuration and Font Family with ClaimCenter

You must register your Apache FOP configuration file and font family with ClaimCenter.

1. Open ClaimCenter Studio and search for `config.xml`.
2. Find the following parameters in `config.xml` and set them accordingly. The following example registers the IPA Gothic font family with ClaimCenter.

Parameter	Description	Example value
`PrintFontFamilyName`	The name of the font family for the custom fonts as defined in your FOP configuration file.	IPA Gothic
`PrintFOPUserConfigFile`	The fully qualified path to a valid FOP configuration file.	`/usr/local/fop-0.95/fopconfig/`

3. Stop and start the application server so that these changes take affect.

Step 6: Test Your Configuration

After you perform the listed configuration steps, Guidewire recommends that you test that you are able to create and print a PDF that uses the correct font. To test the Apache FOP configuration, you must have a ClaimCenter implementation that supports the Japanese locale.

chapter 4

Localizing ClaimCenter String Resources

This topic discusses the different ways you can localize these different kinds of string resources that ClaimCenter displays in the application user interface:

- **Display keys** – Strings to display as field and screen labels and interactive error messages
- **Typecodes** – Strings to display as choices in drop-down lists
- **Script parameter descriptions** – Strings to display as descriptions of script parameters
- **Workflow step names** – Strings to display as individual step names in workflow processes

This topic includes:

See also

- “Localizing Guidewire Workflow” on page 51

Understanding String Resources

Guidewire designs ClaimCenter to use string resources for the following:

- **Display Keys** – Strings to display as field and screen labels and interactive error messages
- **Typecodes** – Strings to display as choices in drop-down lists
- **Script Parameter Descriptions** – Strings to display as descriptions of script parameters

- **Workflow Step Names** – Strings to display as individual step names in workflow processes

You can extract these string resources from ClaimCenter so you can easily localize them separately from other application resources.

Display Keys

ClaimCenter stores as *key/value* pairs the U.S. English string resources from which it generates field and screen labels and interactive error messages in the user interface. Guidewire calls these particular key/value pairs *display keys*. You specify the key/value pair of a display key in standard Java properties syntax, for example:

```
Admin.Workload.WorkloadClassification.General = General
```

ClaimCenter stores the key/value pairs for display keys in a file called `display.properties`. In the base configuration, ClaimCenter contains a single copy of `display.properties`, with string resources in U.S. English. In Studio, you can find the base copy of `display.properties` in the following location, starting from the **Project** window:

configuration → **config** → **Localizations** → **en_US**

If you install a non-U.S. English language module, the installer adds a version of `display.properties` localized for that language.

See also

- "Localizing Display Keys" on page 37.
- "Installing Display Languages" on page 20.

Typecodes

ClaimCenter stores as *key/value* pairs the U.S. English string resources from which it displays choices and choice descriptions in drop-down lists in the user interface. Guidewire calls these particular key/value pairs *typecodes*. You specify the key/value pairs for the name and description of a typecode as a couplet in standard Java properties syntax, for example:

```
TypeKey.CoverageType.CPBldgCov = Building Coverage
TypeKeyDescription.CoverageType.CPBldgCov = Building Coverage
```

ClaimCenter stores the key/value pairs for typecodes in a file called `typelist.properties`. In the base configuration, ClaimCenter contains a single copy of `typelist.properties`, with string resources in U.S. English. In Studio, you can find the base copy of `typelist.properties` in the following location, starting from the **Project** window:

configuration → **config** → **Localizations** → **en_US**

If you install a non-U.S. English language module, the installer adds a version of `typelist.properties` localized for that language.

See also

- "Localizing Typecodes" on page 39.
- "Installing Display Languages" on page 20.

Workflow Step Names

In ClaimCenter, it is possible to provide localized versions for the names of individual steps in a workflow process. It is also possible to set a specific language and set of regional formats on each workflow.

See also

- "Localizing Guidewire Workflow" on page 51
- "Localizing Workflow Step Names" on page 52

Script Parameter Descriptions

File `display.properties` also contains key/value pairs for all script parameters and their descriptions. ClaimCenter stores the script parameter description in a display key of the form `ScriptParameter + <Parameter Name>`, for example:

```
ScriptParameter.InitialReserve_AutoGlassVehicleDamage = ...
```

It is possible to localize this display key in the same manner as you localize any other display key.

ClaimCenter displays the script parameter description in the ClaimCenter **Administration** tab, **Script Parameters** page. To see the parameter description, first select an individual script parameter.

See also

- "Localizing Script Parameter Descriptions" on page 40

Exporting and Importing String Resources

ClaimCenter provides the ability to export all string resources to an external file, including the following:

- **Display Keys** – Strings to display as field and screen labels and interactive error messages
- **Typecodes** – Strings to display as choices in drop-down lists
- **Script Parameter Descriptions** – Strings to display as descriptions of script parameters
- **Workflow Step Names** – Strings to display as individual step names in workflow processes

By exporting and importing string resources, you can make all of your translations directly in a single file. ClaimCenter provides separate commands for exporting and importing string resources.

Command	See...
`gwcc export-l10ns [-Dexport.file] [-Dexport.locale]`	"Using the Export Command Line Tool" on page 35
`gwcc import-l10ns [-Dimport.file] [-Dimport.locale]`	"Using the Import Command Line Tool" on page 36

The commands provides parameters to let you specify the locations of the export and import file and a language-specific set of string resources to export and import. The export and import files are in the format of Java properties files.

See also

- To learn how to use Java property files correctly in ClaimCenter, see "Properties Files" on page 393 in the *Gosu Reference Guide*.

Using the Export Command Line Tool

Guidewire provides a command line tool to manually export certain string resources from ClaimCenter. The command exports the following strings resources as name/value pairs:

- Display keys
- Typecodes
- Script parameter descriptions
- Workflow step names

The export file organizes the strings into translated and non-translated groups. The command provides parameters to let you specify the location of the export file and a language-specific set of string resources to export.

To run the export tool

1. Ensure that the application server is running.
2. Navigate to your application installation `bin` directory, for example:
   ```
   ClaimCenter/bin
   ```
3. Run the following command:
   ```
   gwcc export-l10ns -Dexport.file="targetFile" -Dexport.locale=localeNode
   ```

-Dexport.file Command Parameter

Command line parameter `-Dexport.file` specifies the name of the file (`targetFile`) into which ClaimCenter exports the resource strings. You must add the file extension to the file name. By default, ClaimCenter places the export file in the root of the installation directory:

- To leave the export file in the same location, enter only the name of the file to export.
- To place the file in a different location, enter either an absolute path or a relative path to the file from the root of the installation directory.

-Dexport.locale Command Parameter

Command line parameter `-Dexport.locale` specifies the target locale node (`localeNode`) for the translated strings. The locale node name must match a ClaimCenter locale that exists in the `LanguageType` typelist, for example: `fr_FR` or `ja_JP`.

See also

"Using the Import Command Line Tool" on page 36

Using the Import Command Line Tool

Guidewire provides a command line tool to import localized string resources that you previously exported from ClaimCenter. The command imports the following string resources as key/value pairs:

- Display keys
- Typecode names and descriptions
- Script parameter names
- Workflow step names

The command provides parameters to let you specify the location of the import file and a language-specific set of string resources to import.

To run the import tool

1. Ensure that the application server is running.
2. Navigate to your application installation `bin` directory, for example:
   ```
   ClaimCenter/bin
   ```
3. Run the following command:
   ```
   gwcc import-l10ns -Dimport.file="sourceFile" -Dimport.locale=localeNode
   ```

-Dimport.file Command Parameter

Command line parameter `-Dimport.file` specifies the file (`sourceFile`) that contains the translated resource strings. It must be in the same format as an export file from ClaimCenter. By default, ClaimCenter places the export file in the root of the installation directory:

- If you leave the import translation file in the same location, then you need enter only the name of the file to import.

- If you move the translation file to a different location, then you must enter an absolute or relative path to the file from the root of the installation directory.

-Dimport.locale Comand Parameter

Command line parameter `-Dexport.locale` specifies the name of locale node (`localeNode`) into which to place the translated strings. The locale node name must match a ClaimCenter locale that exists in the `LanguateType` typelist, for example: `fr_FR` or `ja_JP`.

See also

“Using the Export Command Line Tool” on page 35

Localizing Display Keys

ClaimCenter initializes the display key system as it scans all locale nodes in all modules for display key property files (`display.properties` files). See “Properties Files” on page 393 in the *Gosu Reference Guide* for a discussion of the use of property files in Guidewire ClaimCenter.

In the base configuration, ClaimCenter contains a single `display.properties` file, in the following location (starting from **Project**):

configuration → **config** → **Localizations** → **en_US**

The **Localizations** node contains multiple folders with locale-identifier names (**de_DE**, for example), each of which can also contain–after configuration–additional properties files. In addition, any language module that you install also contains property files. See “Display Keys” on page 34 for more information on display keys and string resources in general.

It is possible to provide translated display keys in any of the following ways:

Translate display keys by...	To learn how, see...
Using the Studio **Display Keys** editor	“Localizing Display Keys Using the Display Keys Editor” on page 38
Using the display key import and export tools	“Exporting and Importing String Resources” on page 35

Generating the Master List of Display Keys

Using the locale property files, ClaimCenter generates a master list of display keys. For each property file, ClaimCenter loads the display keys and adds each display key to the master list under the following circumstances:

1. The master list does not already contain the display key.
2. The master list already contains the display key, but, the display key in the master list has a different number of arguments than the display key to add. If this is the case, then ClaimCenter logs a warning message noting that it found a display key value with different arguments in different locales. For example:
   ```
   Configuration Display key found with different argument lists across locales: Validator.Phone
   ```

As ClaimCenter creates the master display key list, it scans the application locale folders in the following order for copies of file `display.properties`:

- The application default locale folder (as set by configuration parameter `DefaultApplicationLanguage`)
- All other locale folders configured for use by the server
- The Guidewire default locale folder (`en_US`)
- All remaining locale folder

After ClaimCenter creates the master list of display keys, the application checks the display keys for the default locale against the master list. ClaimCenter then logs as errors any display keys that are in the master list but missing from the default application locale. For example:

```
ERROR Default application locale (en_US) missing display key: Example.Display.Key
```

As the error message returns the display key name, you can use that name to generate a display key value in the correct locale folder.

Localizing Display Keys Using the Display Keys Editor

It is possible to enter a localized version of a display key directly in the Studio **Display Keys** editor. To access the editor, first, within Studio, navigate to the **Localizations** node. Expand the locale node for the target language and open the `display.properties` file in that folder. You can also use key command CTRL+SHIFT+N to find a properties file for a specific locale.

As indicated, it is possible to enter translated strings directly into the **Display Keys** editor. However, this process can be lengthy, tedious, and error-prone. Guidewire does not recommend this approach if you have a large number of translated strings to enter. Instead, use the export and import commands detailed in “Exporting and Importing String Resources” on page 35 to import translated strings into Studio.

Identifying Missing Display Keys

Guidewire provides a display key difference tool that does the following:

- It compares each locale configured on the server against the master display key list.
- It generates a file that contains a list of any missing keys.

To generate a display key difference report, run the following command from the application `bin` directory:

```
gwcc displaykey-diff
```

The `displaykey-diff` tool creates a new build directory under the application root directory, for example:

```
ClaimCenter/build
```

If the tool detects that an installed language has missing display keys:

- The tool creates a subdirectory for that language using the locale code for that language to name the subdirectory.
- The tool populates that subdirectory with a `display.properties` file containing the list of missing keys.

Each `display.properties` file contains a list of display keys that are in the master list but not in that locale node. The format of the file is exactly the same as the display key configuration files. For example, the following code illustrates the contents of the file for the `en_US` locale:

```
#
# Missing display keys for locale
# en_US
#
Web.QA.I18N.ContactDetail.AddressDetail.City = Suburb
Web.QA.I18N.ContactDetail.AddressDetail.ZipCode = Postcode
```

Note: ClaimCenter does not generate a `display.properties` file for a locale that does not have any missing display keys.

Working with Display Keys for Later Translation

It is possible to create a display key within a specific locale that is not actually localized yet. This display key is simply a placeholder string for a display key that you intend to localize at some point. If you create one of these *to-be-translated* display keys, then Guidewire recommends that you add a suffix of [TRANSLATE] to each display key that you create as a placeholder. For example:

Actions [TRANSLATE]

The suffix can be any string that is meaningful. But, use the same string in all cases as it makes it easy to find the placeholder display keys. Using a string such as [TRANSLATE] makes it easy to see the string in the ClaimCenter interface. It also makes it easy for users to understand that the display key has not yet been translated.

It is important to use the same tag for all of the placeholder strings so you do not miss any during a search.

Localizing Typecodes

You can provide localized typecodes for a typelist in the following ways:

- **Using the Typelist Localization Editor** – Enter localized values for individual typecodes directly through the Typelist editor.
- **Using the gwcc Export and Import Commands** – Use the commands to localize all typecodes through a single text file.

Using the Typelist Localization Editor

You can localize typecodes in the editor only for languages with which you configured ClaimCenter.

To localize typecodes by using the Typelist Localization editor in Studio

1. Type CTRL+SHIFT+N in the Studio **Project** window and enter the name of the typelist that you want to localize.
2. Select a typecode for which you want to provide a localized version.
3. Click the **Localization** link in the bottom right-hand corner of the Typelist editor.
4. Find the group with the locale ID for which you want to provide localized strings
5. Enter localized strings for any or all of the following:
    - **name** – The natural language name associated with this typecode.
    - **desc** – The description of this typecode.

Using the gwcc Export and Import Commands

You use the `gwcc` import and export commands to localize typecodes for all typelists in a single file.

See also

- “Using the Export Command Line Tool” on page 35
- “Using the Import Command Line Tool” on page 36

Setting Localized Sort Orders for Localized Typecodes

It is possible to set the sort order for the typecodes in a typelist. ClaimCenter determines the typecode sort order from a file named after the typelist, with a file extension of `.sort`. ClaimCenter stores the sort order information, by language, in the typelist table.

A typical use for a sort order file is to provide a sort order for Japanese provinces, which customarily are in order from North to South (Hokkaido, Aomori, Iwate, …).

Guidewire does not provide any sort order files in the base configuration. You must place any `.sort` file that you create in the appropriate **Localizations** locale folder. For example, for Japanese, place the file in the following location in Studio:

configuration → **config** → **Localizations** → **ja_JP**

IMPORTANT Any change that you make to a typelist sort order file triggers a database upgrade.

Accessing Localized Typekeys from Gosu

Gosu provides three String representations that you can use for typekeys.

Typekey property	Description
`typelist.typekey.Code`	String that represents the typecode
`typelist.typekey.DisplayName`	Localized language version of the entity name
`typelist.typekey.UnlocalizedName`	Name listed in the data model

For example, to extract localized information about a typekey, you can use the following:

```
var displayString = myTypekeye.DisplayName
```

The following code is a more concrete example.

```
print(AddressType.TC_BUSINESS.DisplayName)
```

It is important to understand that the display key reference acts more as a function call rather than as a value. If the language setting for the user changes, then the display key value changes as well. However, the value stored in `displayString` does not automatically change as the language changes.

Localizing Script Parameter Descriptions

You can localize the descriptions of script parameters. ClaimCenter displays script parameter descriptions in the **Administration** tab, **Script Parameters** screen.

To localize a script parameter description, you must add a display key for the script parameter to file `display.properties` in the localization folder for the target language. Display keys for script parameter descriptions must begin with `ScriptParameter` and follow standard Java property syntax:

```
ScriptParameter.<parameterName> = localizedParameterDescription
```

For example:

```
ScriptParameter.InitialReserve_AutoGlassVehicleDamage = Initial amount to reserve for auto glass damage
```

See also

- “Localizing Display Keys” on page 37

Localizing Gosu Error Messages

Similar to the key/value paris stored in `display.properties`, ClaimCenter stores the string resource that Gosu errors generate in file `gosu.display.properties`. In the base configuration, Guidewire provides only a single English-language version of this file, in the following location in Studio:

configuration → **config** → **Localizations** → **en_US**

File `gosu.display.properties` provides the following key/value pairs, for example:

```
ARRAY = Array
BEAN = Bean
BOOLEAN = Boolean
DATETIME = DateTime
FUNCTION = Function
IDENTIFIER = Identifier
```

```
METATYPENAME = Type
NULLTYPENAME = Null
NUMERIC = Number
OBJECT_LITERAL = ObjectLiteral
STRING = String
MSG_BAD_IDENTIFIER_NAME = Could not resolve symbol for : {0}
...
```

If you install a new language module, that language module contains a translated version of file `gosu.display.properties` in the target language.

chapter 5

Localizing ClaimCenter with Studio

This topic covers how you localize certain types of data using Guidewire Studio.

This topic includes:

Localizing Rule Set Names and Descriptions

In Studio, it is possible to show rule names, rule set names, and rule set descriptions in a language other than English. To display a rule name, or a rule set name and description in another language, you can translate these items in the definition file for that rule or rule set.

Within Studio, ClaimCenter shows the rule sets in a hierarchical structure, thusly:

Rule set category → *Rule set* → *Rule*

Within the application directory structure, Guidewire stores rule-related files in the following location:

```
modules/configuration/config/rules/...
```

The following list describes the various rule file types and what you can localize in each file type.

File type	Rule type	Translatable unit	Example
`.grs`	Rule set	Rule set name	`@gw.rules.RuleName("`*`Some translated rule set name`*`")`
		Rule set description	`@gw.rules.RuleSetDescription("`*`Some translated description`*`")`
`.gr`	Rule	Rule name	`@gw.rules.RuleName("`*`Some translated rule name`*`")`

You modify the rule definition file directly in the operating system directory structure. However, only modify rule definition files in the `modules/configuration/config/rules` directory.

You can only view a single translated version of a rule set name or description in Studio.

Setting a Language for a Block of Gosu Code

It is possible to set a specific locale in any Gosu block (in Gosu code) by wrapping the Gosu code in any of the following methods.

```
gw.api.util.LocaleUtil.runAsCurrentLanguage(alternateLocale, \ -> { code } )
gw.api.util.LocaleUtil.runAsCurrentLocale(alternateLocale, \ -> { code } )
gw.api.util.LocaleUtil.runAsCurrentLocaleAndLanguage(alternateLocale, alternateLocale, \ -> { code } )
```

The method parameters take the following form:

Parameter	Description
`alternateLocale`	The locale or language to use in the form of an `ILocale`, for example: `gw.i18n.ILocale.fr_FR`
`\ -> { code }`	A Gosu block as a `GWRunnable` object—the Gosu code to run in a different locale or language

For example:

```
uses gw.i18n.ILocale
gw.api.util.LocaleUtil.runAsCurrentLocaleAndLanguage(ILocale.FR_FR, ILocale.FR_FR, \ -> { ... } )
```

Example

Suppose that you want to format a date for a particular language. For example, you want to represent a date in the format `yyyy-dd-MM` (as is typical for French dates). You can use the Gosu `DateUtil` method to create a `GWLocale` object from the date. The code block then uses the `runAsCurrentLocale` method to format the date using French regional formatting conventions.

```
uses gw.api.util.LocaleUtil
uses gw.i18n.ILocale
uses gw.api.util.DateUtil
uses java.util.Date

var formattedDate : String

LocaleUtil.runAsCurrentLocale(ILocale.FR_FR,
        \ -> { formattedDate = DateUtil.currentDate().format("long")})
```

Additional Useful Methods on gw.api.util.LocaleUtil

In addition to the `runAsCurrentLocale` method, `gw.api.util.LocaleUtil` contains a number of other methods useful in working with localization. Some of the more important ones are:

Method	Returns...
`canSwitchLanguage`	Boolean `true` if the current user is allowed to switch to a different language
`canSwitchLocale`	Boolean `true` if the current user is allowed to switch to a different locale
`getAllLanguages`	List of all available languages as defined in the `LanguageType` typelist
`getAllLocales`	List of all available languages as defined in the `LocaleeType` typelist
`getCurrentLanguage`	Current language, which can be based on the user language setting
`getCurrentLocale`	Current locale, which can be based on the user language setting
`getCurrentUserLanguage`	Language set for the current user
`getCurrentUserLocale`	Locale set for the current user
`getDefaultLanguage`	Language associated with the default locale (as set by `DefaulApplicationtLocale`)
`getDefaultLocale`	Applications default locale as set by configuration parameter `DefaulApplicationtLocale`
`getLanguageLabel`	Language as a `String` value, given a language type
`getLocaleLabel`	Locale as a `String` value, given a locale type
`toLanguage`	Language type, given a locale
`toLocale`	Locale, given a language type

Localizing Administration Tool Argument Descriptions

Guidewire provides a set of administration tools defined in Gosu. You use these tools, for example, to control server behavior, to load data, and to generate tools for use in the Guidewire configuration environment.

These command line tools derive their arguments from properties on a class that ClaimCenter initializes at the start of each tool, depending on the tool. ClaimCenter stores these files in the following location:

```
ClaimCenter/admin/src/gw/cmdline/util/...
```

In the base configuration, this directory contains a number of command line argument files, including the following:

- `ImportToolArgs.gs`
- `SystemToolArgs.gs`
- `TableImportArgs.gs`
- `WorkflowToolsArgs.gs`
- `ZoneImportToolArgs.gs`

Note: For a full list of the command line tools available in this directory, see “ClaimCenter Administrative Commands” on page 175 in the *System Administration Guide*.

Each of these Gosu classes contains a set of static properties with associated comments. By default, ClaimCenter derives the description of each command line argument from the comments in the file associated with that administration tool. However, it is possible to create a separate property file, with localized name/value pairs of the property descriptions and associate that file with a tool. ClaimCenter then reads the associated property file and displays localized versions of the tool argument descriptions.

For example, in `SystemToolsArgs`, you see the following definition for the `server` command line argument:

```
/**
* The password to use
*/
@ShortName( "password" )
@LongName( null )
static var _password : String as Password
```

Suppose that you want to translate the description of the `Password` argument for a French speaker. To do so, first create the following file:

```
ClaimCenter/admin/src/gw/cmdline/util/SystemToolArgs_fr_FR.properties
```

Then, populate the file that you just created with the following text:

```
SystemToolsArgs.Password=Le mot de passe à utiliser
```

In this way, it is possible to localize the property name descriptions (which are the command line argument descriptions) in these files. To do so, you need to do the following:

- Create a properties file and populate it with translated key/value pairs.
- Name the file using the standard Java convention for differentiating between various localizations.
- Place the properties file in the same directory as the class file that it modifies.

IMPORTANT Guidewire ClaimCenter displays the localized descriptions for the administration tools for the operating system locale, not the locale set for Guidewire ClaimCenter.

To localize the command line argument descriptions

1. Create a properly translated properties file, with a name/value pair for each property in the class for which you are providing localization.
    - *Name* is the property in class `system_tools`, for example, `SystemToolsArgs.Server`.
    - *Value* is the properly localized description of that property name.

The property file must have a name/value pair for each property in the class for which you want to provide localization. If there is no entry for a given property, ClaimCenter uses the default documentation from the comment on the property in class.

2. Name the property file with the same name as the tool class, using the standard Java conventions for differentiating between various localizations. For example, if creating a localized `SystemToolArgs` properties file for German, use the following:

   ```
   SystemToolsArgs_de_DE.properties
   ```

3. Place the translated properties file in the same directory as the tool class, which is:

   ```
   ClaimCenter/admin/src/gw/cmdline/util/...
   ```

To access the command line tools in a chosen locale

Guidewire ClaimCenter displays the localized descriptions for the command line tools for the operating system locale, not the locale set for Guidewire ClaimCenter. Thus, to run the command line tools in a specific locale, do one of the following:

- Set your operating system to the chosen locale.
- Set the command line window to the chosen locale.

The following steps illustrate how to set the locale of a command line window using the `GOSU_OPTS` command.

1. Open a command line window and navigate to the following directory:

   ```
   ClaimCenter/admin/bin
   ```

2. Enter the following command, replacing *<locale>* with the exact same locale that you used to name your localization property file.

   ```
   ClaimCenter/admin/bin/set GOSU_OPTS=-Duser.language=<locale>
   ```

   For example, if you create a property localization file named `SystemToolsArgs_de_DE.properties`, then enter the following:

   ```
   ClaimCenter/admin/bin/set GOSU_OPTS=-Duser.language=de_DE
   ```

3. Run your administration tool.

You can view a list of arguments and descriptions by entering the tool name without any arguments. For example, if you enter the following, ClaimCenter displays a list of the arguments that you can use with the system tools command:

```
ClaimCenter/admin/bin/system_tools
```

See also

- “Generating Java and SOAP API Libraries” on page 96 in the *Installation Guide*
- “ClaimCenter Administrative Commands” on page 175 in the *System Administration Guide*

chapter 6

Localizing Administration Data

You can localize shared administration data through the use of localized database columns. For example, ClaimCenter uses activity patterns to create new activities. A localization column lets users see activity names in their preferred languages.

This topic includes:

Understanding Administration Data

Guidewire refers to certain types of application data to as *administration data* or admin data, for short. For example, activity patterns are administration data. For selected fields in administration data, ClaimCenter stores localized or translated values directly in the application database.

You enter localized admin data directly within Guidewire ClaimCenter, generally in a **Localization** table that is visible at the bottom of a screen. This table contains a row for each enabled language in the application and fields for each element on that screen that you can localized. You must have more than one installed and configured language for ClaimCenter to show the **Localization** table.

Localized Columns in Entities

To accommodate localized values for shared administration data or any entity data, ClaimCenter lets you designate a column as containing localized values in the database. To configure an entity to store localized values for a column, add a `<localization>` element as a child of the `<column>` element in the metadata definition of the entity.

Adding a `<localization>` element to a column triggers the creation of a *translation table* during the next database upgrade. A translation table stores localized values for a column for every language other than the default application language. The column itself stores values for the default application language.

The `<localization>` element has the following form.

```
<?xml version="1.0"?>
<entity ... >
  ...
  <column ... >
    <localization tableName="<mainEntityNameAbbrevation>_<columnNameAbbreviation>_L10N"/>
  </column>
  ...
</entity>
```

WARNING The `<localization>` element has the required attribute `tableName`, which is the table name of the translation table. Do not exceed a length of 16 characters for `tableName`. If you do, the application server will not start.

See also

- "Translation Tables in the Database" on page 49

Localization Attributes

Guidewire provides several specialized attributes on the `<localization>` subelement on `<column>` that impact the use of the element. The following list describes each attribute:

Attribute	Type	Description
`unique`	`Boolean`	If you set this attribute to `true`, ClaimCenter enforces that for each language the values are unique and there are no duplicates.
`nullok`	`Boolean`	If you set this attribute to `true`, ClaimCenter flags missing entries that it finds during a database consistency check. For example, suppose that ClaimCenter defines two or more languages: • ClaimCenter stores the values for the default application language in the main database table of the entity. • ClaimCenter stores the values for additional languages in a separate translation table. During a consistency check, ClaimCenter flags entries in the main database table for the default language if corresponding entries for additional languages cannot be found in the translation table. Entries flagged as missing additional languages are warnings only. A missing language value does not prevent the database server starting.

See also

- "Checking Database Consistency" on page 42 in the *System Administration Guide*

Example Localization Column on the Activity Pattern Entity

The following metadata definition of the `ActivityPattern` entity configures the `Description` column for localization in the base configuration. Thus, the `Description` column can store localized values for each configured language.

The `ActivityPattern` entity looks similar to the following.

```
<?xml version="1.0"?>
<entity xmlns="http://guidewire.com/datamodel"
        admin="true"
        desc="An activity pattern is a template for an activity. "
        entity="ActivityPattern"
        exportable="true"
        extendable="true"
        javaClass="com.guidewire.pl.domain.activity.ActivityPatternBase"
        platform="true"
        table="activitypattern"
        type="retireable">
  <implementsEntity name="DecentralizedEntity"/>
  <column name="Description" desc="Description of the activity pattern." type="mediumtext">
    <localization tableName="actpat_desc_l10n"/>
  </column>
   ...
</entity>
```

Translation Tables in the Database

ClaimCenter stores the localized values for columns that have a `<localization>` element in separate translation tables in the database. ClaimCenter generates translation tables automatically. Guidewire recommends that you use the following format for the table name attribute.

```
<localization tableName="<mainEntityNameAbbrevation>_<columnNameAbbreviation>_L10N"/>
```

The length of the table name must not exceed 16 characters.

For example, the translation table name for the `Subject` column of the `ActivityPattern` entity uses the abbreviations `sbj` and `actpat`, as the following example shows.

```
actpat_sbj_L10n
```

Translation tables include the following columns:

- `Owner` – An integer that represents an ID of the owner
- `Language` – A typekey to the `LanguageType` typelist
- `Value` – A column of type `String`

Translation tables contain localized values for configured languages other than the default application language. The localized column itself contains values for the default application language.

chapter 7

Localizing Guidewire Workflow

This topic discusses localization as it relates to Guidewire workflow.

This topic includes:

Localizing ClaimCenter Workflow

At the start of the workflow execution, the Workflow engine evaluates the language and locale set for the particular workflow. The Workflow engine then uses that language for notes, documents, templates, and similar items associated with the workflow. The specific language and locale that the workflow uses depends on the specific user (`userid`) that executes the workflow code.

However, it is possible to set the workflow locale and the workflow language independently of the default application language and locale. To set either workflow language or locale, click in the background area in the workflow layout view. This action opens the **Properties** area at the bottom of the workflow area. In this **Properties** area, you can enter one of the following:

- A fixed name for the language or locale
- A Gosu expression that evaluates to a valid type for the language or locale

For example, to set a specific workflow locale, use one of the following:

Type	Gosu
Fixed string	`gw.i18n.ILocale.fr_FR`

Variable expression	`gw.api.util.LocaleUtil.toLocale(  thisClaim.Claimant.PrimaryLanguage )`
	`gw.api.util.LocaleUtil.toLocale( Group.Supervisor.Contact.PrimaryLanguage )`

Localizing Workflow Step Names

You can provide translated versions of workflow step names. ClaimCenter displays translated step names in the following locations:

- **Workflow summary** – On the **Find Workflows** search screen, the workflow summary shows the last completed workflow step. Administrative accounts only can access this screen.
- **Workflow log** – On the **Workflow Detail** screen, the log shows all workflow steps.

Users can see translated step names in ClaimCenter by using the **Language** submenu on the **Options** menu.

You can provide translated workflow steps names by following one of these procedures:

- Entering a Translated Workflow Step Name in Studio
- Exporting Workflow Steps Names as String Resources for Translation

See also

“User Selection of Language and Regional Formats” on page 11

Entering a Translated Workflow Step Name in Studio

To enter a translated workflow step name in Studio

1. Open the target workflow in the Studio **Workflow** editor.
2. Select the workflow step to localize.
3. In the Properties tab at the bottom of the screen, select the **Step Name Localizations** tab.
4. Click **Add** and enter the translated step name inn the target language.

Exporting Workflow Steps Names as String Resources for Translation

To export workflow step names as string resources for translation

1. Export the ClaimCenter string resources for a particular locale using the following `gwcc` command:
   ```
   gwcc export-l10ns ...
   ```
   In the command, you must provide a target file name and specify from which locale node to export the string resources.
2. Translate the appropriate workflow step names into the target language.
3. Import the translated strings resources back into ClaimCenter using the following `gwcc` command:
   ```
   gwcc import-l10ns ...
   ```
   In the command, you must provide a source file name and specify into which locale folder to import the string resources.

See also

- For command parameters, “Exporting and Importing String Resources” on page 35.
- To understand their format and syntax, “Properties Files” on page 393 in the *Gosu Reference Guide*.

Creating a Locale-Specific Workflow SubFlow

You can create a child workflow, or subflow, in Gosu by using the following methods on `Workflow`. Each method handles the locale of the subflow differently.

Method	Description
`createSubFlow`	Creates a child subflow synchronously, meaning that the Workflow engine starts the subflow immediately upon method invocation. The new subflow automatically uses the default application locale, not the locale of the parent workflow. Thus, if you set the locale of the parent workflow to be different from the default application locale, the subflow does not inherit that locale.
`createSubFlowAsynchronously`	Creates a child subflow asynchronously, meaning that the Workflow engine does not start the subflow until it finishes executing all code in the Gosu initialization block. Again, the subflow uses the default application locale, not the locale set for the workflow itself. As the Workflow engine executes all the Gosu code in the block before starting the subflow, it is possible to set the locale of the subflow before the workflow starts.
`instantiateSubFlow`	Creates a child subflow, but does not start it. If desired, you can modify the subflow that the method returns before you start the subflow.

To create a subflow that inherits the locale of the parent workflow

1. Define a workflow that has the `LanguageType` property. See “Creating New Workflows” on page 409 in the *Configuration Guide* for information on how to create a new workflow with a `LanguageType` property.
2. Set the locale for this subflow so that it uses your desired language. See “Localizing ClaimCenter Workflow” on page 51 for details.
3. Instantiate the subworkflow using the `instantiateSubFlow` method rather than the `createSubFlow` method.
4. Set the `LanguageType` property on the instantiated subflow to the locale of the parent workflow.
5. Start the subflow, using one of the workflow start methods described at “Instantiating a Workflow” on page 412 in the *Configuration Guide*.

See also

“Workflow Subflows” on page 418 in the *Configuration Guide*

Localizing Gosu Code in a Workflow Step

It is possible to localize Gosu code that you add to any workflow step (an **Enter Script** block, for example). To do so, wrap the Gosu code in the following method.

```
gw.api.util.LocaleUtil.runAsCurrentLocale(alternateLocale, \ -> { code } )
```

In the code, set the value of `alternateLocale` to the locale to use for the Gosu code block, for example:

```
gw.api.util.LocaleUtil.runAsCurrentLocale(gw.i18n.ILocale.LANGUAGE_FR_FR, \ -> { code } )
```

See “Setting a Language for a Block of Gosu Code” on page 44 for further examples of how to work with this method.

Other wrapper methods useful for localization include the following:

- `runAsCurrentLanguage`
- `runAsCurrentLocaleAndLanguage`

part II

Regional Formats Configuration

chapter 8

Working with Regional Formats

ClaimCenter lets you configure support for multiple regional formats. Regional formats specify how to format dates, times, numbers, and monetary amounts that users enter and retrieve. Regional formats specify the visual layout of data that has no inherent association with specific countries/regions but for which formats vary by regional convention.

This topic includes:

Understanding Regional Formats in ClaimCenter

The base configuration of ClaimCenter provides a number of folders in **configuration** → **config** → **Localizations**. The localization folders have locale identifiers for names, such as:

- **de_DE**
- **en_US**
- **fr_FR**
- **ja_JP**

These locale folders can contain the following:

- One or more files that define regional formats for specific localities, such as France
- One or more files that configure a specific language, such as French

In the base configuration, only the `en_US` locale folder has a full set of localization files for the English language and for regional formats customary to the U.S.

The following list describes the files that it is possible to find in a **Localization** folder. The **Locale** column lists the locale or locales in which a file exists in the base configuration.

File	Contains...	Locale
Language-related files...		
`collations.xml`	Specialized rules for database sort and search for various languages.	-
`display.properties`	Display keys to use within Guidewire ClaimCenter. The text strings in this file are in the language specified by the name of the containing folder. For example: • In `en_US`, the text strings are in English. • In `fr_FR`, the text strings are in French. The `display.properties` file (as with all property files) is a standard Java properties file. See "Properties Files" on page 393 in the *Gosu Reference Guide* for a discussion of the use of property files in Guidewire ClaimCenter.	`en_US`
`gosu.display.properties`	Display keys used for Gosu error messages. The Gosu compiler displays one of these messages if it encounters a Gosu error condition. The text strings in this file are in the language specified by the name of the containing folder.	`en_US`
`language.xml`	Language definitions to configure in-memory linguistic collation. In the base configuration, only the `de_DE` folder contains a `lanugage.xml` file. ClaimCenter uses this file to set German-language collation rules.	`de_DE`
`typelist.properties`	Typekeys used within ClaimCenter. The text strings in this file are in the language specified by the name of the containing folder.	`en_US`
Format-related files...		

`localization.xml`	Formats and other types of configuration data to use with a specific country. For example:
	• In `en_US`, the file contains configuration information on date, time, number, and currency formats for use in the United States.
	• In `ja_JP`, the file contains configuration information on how to format address information for Japan. It also contains configuration information on the Japanese Imperial Calendar in use in Japan.
	Note: If you do not supply the relevant configuration data in a `localization.xml` file, then ClaimCenter uses the ICU library defaults. See "The International Components for Unicode (ICU) Library" on page 59 for details.
	In the base configuration, Guidewire uses the ICU library defaults for certain locales. Other locales require more specific configuration and thus require a `localization.xml` file.
		`en_US`
		`fr_FR`
		`ja_JP`

Note: For a discussion of how to work with the display keys and typecodes that exist in ClaimCenter, see "Localizing ClaimCenter String Resources" on page 33.

It is possible to add a new locale folder to **Localizations**. To do so, use the contextual right-click menu to create a new folder. You then need to populate the folder with the appropriate files and configuration data.

It is also possible for ClaimCenter to create and populate a new locale-identifier folder as part of a language module installation. In this case, ClaimCenter creates the new folder under **config** → **Localizations** in the language module itself.

ClaimCenter does not prevent you from creating multiple copies of the same locale folder, one in the main **configuration** module and another in language module. If there are multiple copies of a locale folder, then the contents of the folder in the main **configuration** module override any files that exist in any language module.

Property Files

ClaimCenter uses a number of property files. Property files are text files that have `.properties` extension. All ClaimCenter property files use the standard Java format for property files. Property files in use in ClaimCenter include the following, for example:

- `display.properties`
- `gosu.display.properties`
- `typelist.properties`
- ...

See “Properties Files” on page 393 in the *Gosu Reference Guide* for a discussion of the use of property files in Guidewire ClaimCenter.

The International Components for Unicode (ICU) Library

If a locale folder does not contain a `localization.xml` file, which defines regional formats for a locality, ClaimCenter uses the ICU library defaults for that locale. See “The International Components for Unicode (ICU) Library” on page 59 for details of the ICU Library.

If you do not supply the relevant configuration data in a `localization.xml` file (**configuration** → **config** → **Localizations** → **xx_yy**), then by default:

- ClaimCenter uses the ICU library for regional formats for dates, times, numbers, and monetary amounts.
- ClaimCenter uses the ICU library for the Japanese Imperial Calendar.

The International Components for Unicode (ICU) library is an open source project that provides support for Unicode and software globalization. The library, `icu4j`, attempts to maintain API compatibility with the standard Java JDK, but the ICU library often provides significant performance improvements and a richer feature set. The core of the ICU library is the Common Locale Data Repository (CLDR). The CLDR repository is comprehensive repository of locale data.

See also

- For a feature comparison between the ICU library and the Oracle JDK, see `http://site.icu-project.org/`.
- For comparisons of text collation performance, see `http://site.icu-project.org/charts/collation-icu4j-sun`.
- For more information about the CLDR, see `http://cldr.unicode.org/`.

Configuring Regional Formats for a Region

By convention, specify the name and description for typecodes in the `LocaleType` typelist as:

```
country (language)
```

For example, you add a typecode for the Java locale `nl_NL` to configure regional formats used in the Netherlands. Specify the name and description as:

```
Netherlands (Dutch)
```

See also

- “The International Components for Unicode (ICU) Library” on page 59

Setting the Default Application Locale for Regional Formats

The default application locale determines the regional formats for users who have not chosen a personal preference. You must set a value for the default application locale, even if you configure ClaimCenter with a single locale as the choice for regional formats. You set the default application locale with configuration parameter `DefaultApplicationLocale`, in file `config.xml`.

The following example sets the default application locale to `French (Franch)`, which sets the default choice for regional formats.

```
<param name="DefaultApplicationLocale" value="fr_FR" />
```

The value for `DefaultApplicationLocale` must match a typecode in the `LocaleType` typelist. If you set the value of parameter `DefaultApplicationLocale` to a value that does not exist as a `LocaleType` typecode, then the application server refuses to start. In the base configuration, the default application locale is set to `en_US`.

Working with Regional Formats In Gosu

The `gw.api.util.LocaleUtil` class provides the following methods to let you run a block of Gosu code with a specific set of regional formats:

- `runAsCurrentLocale(`*`localeInterface`*`, \ -> {` *`GosuCode`* `} )`
- `runAsCurrentLocaleAndLanguage(`*`localeInterface`*`,` *`localeInterface`*`, \ -> {` *`GosuCode`* `} )`

Typically, you run a block of Gosu code with a specified set of regional formats so that ClaimCenter formats dates, times, numbers, and human names appropriately for the current user. Otherwise, the block of Gosu code uses the regional formats specified by the `DefaultApplicationLocale` parameter in `config.xml`.

The method takes the following parameters.

Parameter	Description
`localeInterface`	A locale interface that represents a language from the `LanguageType` typelist or a set of regional formats from the `LocaleType` typelist.
`\ -> {` *`code`* `}`	A Gosu block as a `GWRunnable` object—the Gosu code to run with different regional formats or a different language

Obtaining a Locale Interface for a Locale Type

To run a block of Gosu code with a specified set of regional formats, you must specify a locale interface from the package `gw.i18n.ILocale`. Use the `gw.api.util.LocaleUtil.toLocale` method to obtain a locale interface object that corresponds to a Gosu typecode in the `LocaleType` typelist.

The following example Gosu code formats today's date according to the user's preferred regional formats. The code first calls `getCurrentUserLocale` to obtain a `LocaleType` object that represents the user's preferred regional formats. Then, the code calls `toLocale` to convert the `LocaleType` object to a corresponding `ILocale` interface. Finally, the code calls `runAsCurrentLocale` runs to run a block of code that prints today's date formatted according the user's preferred date format.

```
uses gw.api.util.LocaleUtil
uses gw.api.util.DateUtil

// Obtain a locale interface for the user's current set of regional formats.
var currentLocaleType = LocaleUtil.getCurrentUserLocale()
var currentLocaleInterface = LocaleUtil.toLocale(currentLocaleType)

// Run a block of Gosu code that prints the current date in the user's preferred regional format
LocaleUtil.runAsCurrentLocale(currentLocaleInterface,
                              \-> {print(DateUtil.currentDate().format("long"))}
                             )
```

See also

- "Gosu Blocks" on page 231 in the *Gosu Reference Guide*

Configuring the Catastrophe Heat Map Locale

Guidewire maps the ClaimCenter locale setting to available locales for Bing maps (used in the catastrophe map) in the following Gosu file:

```
gw.api.heatmap.BingMap.gs
```

In the base configuration, ClaimCenter defines a `HashMap` in this file that contains the following values:

```
var bingLocaleMap : HashMap = new HashMap<String, String>() {
  "en_AU" -> "en-AU",
  "en_CA" -> "en-CA",
  "en_GB" -> "en-GB",
  "en_US" -> "en-US",
  "fr_FR" -> "fr-FR",
  "ja_JP" -> "ja-JP"
}
```

If you add additional locales to ClaimCenter, then you need to modify this file and add the locale to `bingLocaleMap`.

Adding a new locale to `bingLocaleMap` affects only the localizable elements of the catastrophe map itself, meaning the place names and the labels and tooltips that are on the map. You localize the non-map part of the **Catastrophe Search** screen in the same as you localize the rest of ClaimCenter.

To view the current Bing locale

To determine the current locale in use for a Bing map, use the following Gosu:

```
BingMap.getBingMapsLocale()
```

This function returns the closest equivalent to the Bing locale string, or U.S. English, if there is no close equivalent.

To view the list of Bing supported languages

To view the list of Bing supported languages, navigate to the following:

```
http://msdn.microsoft.com/en-us/library/cc469974.aspx
```

chapter 9

Working with the Japanese Imperial Calendar

This topic discusses the Japanese Imperial Calendar and how to configure Guidewire ClaimCenter to access and display Japanese Imperial Calendar dates correctly.

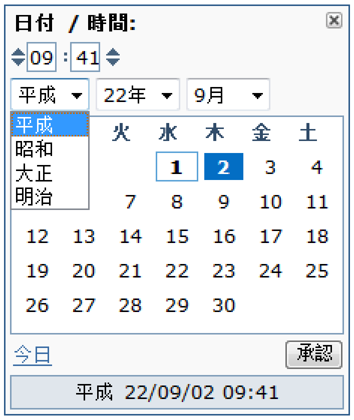
This topic includes:

The Japanese Imperial Calendar Date Widget

The ClaimCenter interface supports the use of a Japanese Imperial calendar date widget. You must enable this feature through configuration. After you enable it, you can specify the default calendar type for the following:

- An entity property
- A Java or Gosu property

The following graphic illustrates the *Date picker* for the Japanese Imperial calendar:



The date widget only displays the four most recent Imperial eras. They are:

- *Heisei*
- *Showa*
- *Taisho*
- *Meiji*

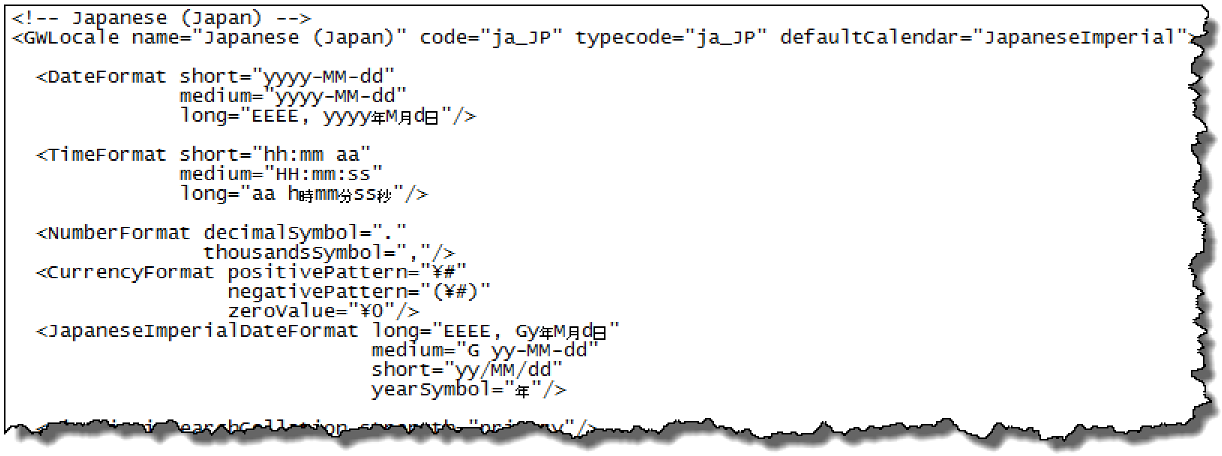
Note: Guidewire does not support the use of an input mask for Japanese Imperial calendar input. Enter date input through the use of the date picker. If you use a copy-and-paste method to enter a date, then you must paste an exact matching string into the date field.

Configuring Japanese Dates

You configure how ClaimCenter displays Japanese Imperial calendar dates by specifying the `<JapaneseImperialDateFormat>` element in `localization.xml`. The `<JapaneseImperialDateFormat>` element is similar to the existing Gregorian `<DateFormat>` element. You need to define the following attributes:

Date format	Description
`long` `medium`	ClaimCenter only uses the `long` and `medium` date formats to display Japanese dates. You cannot use these fields to format user input data.
`short`	ClaimCenter uses the `short` date format to format user date input. Any date input format pattern that you choose must be compatible with the fixed width of the input mask. Guidewire recommends that you use `short=“yy/MM/dd”`.
`yearSymbol`	ClaimCenter uses the Japanese `yearSymbol` as a signal to render the Japanese Imperial calendar date picker.

The following graphic illustrates a Japanese `<GWLocale>` definition along with the associated `<JapaneseImperialDateFormat>` element.



```
<!-- Japanese (Japan) -->
<GWLocale name="Japanese (Japan)" code="ja_JP" typecode="ja_JP" defaultCalendar="JapaneseImperial">

  <DateFormat short="yyyy-MM-dd"
              medium="yyyy-MM-dd"
              long="EEEE, yyyy年M月d日"/>

  <TimeFormat short="hh:mm aa"
              medium="HH:mm:ss"
              long="aa h時mm分ss秒"/>

  <NumberFormat decimalSymbol="."
                thousandsSymbol=","/>
  <CurrencyFormat positivePattern="¥#"
                  negativePattern="(¥#)"
                  zeroValue="¥0"/>
  <JapaneseImperialDateFormat long="EEEE, Gy年M月d日"
                              medium="G yy-MM-dd"
                              short="yy/MM/dd"
                              yearSymbol="年"/>
```

Setting the Japanese Imperial Calendar as the Default for a Region

ClaimCenter provides a way to set a default calendar for a region through the `defaultCalendar` attribute of the `<GWLocale>` element. For example, to set the Japanese Imperial calendar as the default calendar for Japan, add the `defaultCalendar` attribute to the `<GWLocale>` element for `ja_JP` with the value set to `JapaneseImperial`.

```
<GWLocale code="ja_JP" name="Japanese" typecode="ja_JP" defaultCalendar="JapaneseImperial">
```

IMPORTANT Guidewire uses the *International Components for Unicode* (ICU) open source library to format dates according to the Japanese Imperial calendar. The ICU library specifies formats according to the historical Imperial calendar. Contemporary changes to the calendar, such as the start of a new era, requires an updated ICU library. If such an event occurs, contact Guidewire support for details on how to upgrade to a newer version of the ICU library.

See also

- For information on how to set up a region for purposes of the Japanese Imperial calendar, see "Working with Regional Formats" on page 57.

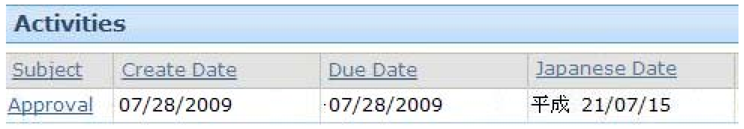
Setting a Field to Always Display the Japanese Imperial Calendar

Suppose, for example, that you want to add an additional field to the `Activity` object that uses the `japaneseimperialdate` data type.

1. First, open `Activity.etx`,which exists in **Data Model Extensions** → **extensions**.
2. Enter something similar to the following:

```
<?xml version="1.0"?>
<extension xmlns="http://guidewire.com/datamodel" entityName="Activity">
  <column desc="For use with Japanese Imperial calendar fields"
          name="JICDate"
          type="japaneseimperialdate">
  </column>
</extension>
```

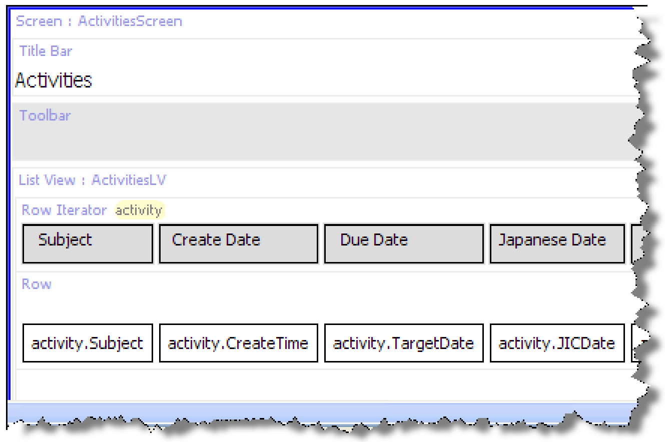
To be useful, you need to use this field to display a value in the Japanese Imperial calendar format. For example, in the **Activities** screen, you can add a **Japanese Date** field.



Activities

Subject	Create Date	Due Date	Japanese Date
Approval	07/28/2009	07/28/2009	平成 21/07/15

In the `Activities` PCF file, it looks similar to the following:



IMPORTANT You must configure your installation for the Japanese locale in order to display a date in Japanese Imperial calendar format. Otherwise, regardless of the calendar setting, you see only Gregorian dates.

Setting a Field to Conditionally Display the Japanese Imperial Calendar

Suppose, for example, that you want to treat a field as either a Gregorian date or a Japanese Imperial calendar date depending on certain factors. In this case, it is possible to write a datatype annotation that causes ClaimCenter to display certain fields in different formats in different situations. For example (as described in “Working with Japanese Imperial Dates in Gosu” on page 68), suppose that you want to do the following:

- If the user's locale is `ja_JP` and `Policy.PolicyType != "CALI"`, then ClaimCenter is to show these fields using the Gregorian calendar.
- If the user's locale is `ja_JP` and `Policy.PolicyType == "CALI"`, then ClaimCenter is to show these fields using the Japanese calendar.

ClaimCenter provides several different ways to accomplish this date formatting. For example, you can:

- Annotate an entity field (database column)
- Annotate a Gosu property

To annotate an entity field

1. First, configure the data type for the entity field in an extension file. For example:

```
<column name="JICDate" type="japaneseimperialdate"/>
```

   See the discussion in “Setting a Field to Always Display the Japanese Imperial Calendar” on page 65 for details if necessary.

2. Define a `PresentationHandler` class for the data type. For example:

```
class JapaneseImperialDateDataTypeDef implements IDataTypeDef {

  construct() { }

  override property get PresentationHandler() : IDataTypePresentationHandler {
    return new JapaneseImperialCalendarPresentationHandler()
  }
  ...
}
```

   If the class that you create implements an interface (in this case, `IDataTypePresentationHandler`), then Studio highlights the interface name in red and marks the class as invalid. This is because you need to provide implementations for all methods and properties in the interface. If you place your cursor in the class definition line, and press ALT-ENTER, Studio automatically inserts the required method and property bodies for you. You then need to define these properties and methods to suit your business needs.

   For an example of how to implement a presentation handler class, see “Defining a New Tax Identification Number Data Type” on page 263 in the *Configuration Guide*. For sample code for a JIC presentation handler, see “Working with Japanese Imperial Dates in Gosu” on page 68.

3. Implement the date presentation handler for the data type. You can make the implementation logic dynamic based on the entity context. For example:

```
class JapaneseImperialCalendarPresentationHandler implements IDatePresentationHandler {

  construct() { }

  override function getCalendar( ctx: Object, prop: IPropertyInfo ) : DisplayCalendar {
    /** if the "ctx" object is a policy and the policy is "cali" type and the "prop"
      * is the effectivedate field ...
      **/
    if (...)
      return JAPANESEIMPERIAL }
    else
      return GREGORIAN
  }
  ...
}
```

To annotate a Gosu property

1. Annotate the data type for the Gosu property. For example:

```
@DataType("japaneseimperialdate")
property get JICDate() : Date {
  return _bean.DateField1
}
```

2. Define a `PresentationHandler` class for the data type in a similar fashion to that described previously in step 2 of *To annotate an entity field*.

3. Implement the date presentation handler class for the data type in a similar fashion to that described previously in step 3 of *To annotate an entity field*.

See Also

- For a working example of how to implement the `IDataTypeDef` interface, see “Defining a New Tax Identification Number Data Type” on page 263 in the *Configuration Guide*. Specifically, see “Step 2: Implement the IDataTypeDef Interface” on page 264 in the *Configuration Guide*.
- For sample code for a JIC presentation handler, see “Working with Japanese Imperial Dates in Gosu” on page 68.

Working with Japanese Imperial Dates in Gosu

Guidewire ClaimCenter provides the ability to create an object extension of type `japaneseimperialdate` based on `java.util.Date`. You can use this data type to do the following:

- To set a calendar field value so that it always displays values in Japanese Imperial calendar style.
- To set a calendar field value so that it displays values in Japanese Imperial calendar style depending on a data field. This requires additional configuration.

For example, consider the following two fields:

- `Policy.EffectiveDate`
- `Policy.ExpirationDate`

Now consider the following cases:

Locale	Policy.PolicyType	Description
`en_US`	–	If `Policy.PolicyType` is `null` (unspecified), then ClaimCenter displays all date fields in the Gregorian calendar in all situations
`ja_JP`	–	If `Policy.PolicyType` is `null` (unspecified), then ClaimCenter displays all date fields in the Japanese Imperial calendar in all situations
`en_US`	=="CALI"	If `Policy.PolicyType = "CALI"`, then ClaimCenter displays `Policy.EffectiveDate` and `Policy.ExpirationDate` in the Japanese Imperial calendar and all other date fields in the Gregorian calendar.
`ja_JP`	!="CALI"	If `Policy.PolicyType != "CALI"`, then ClaimCenter displays `Policy.EffectiveDate` and `Policy.ExpirationDate` in the Gregorian calendar and all other date fields in the Japanese Imperial calendar.

Note: CALI (*Compulsory Auto Liability Insurance*) is a type of automobile policy in Japan that every driver must carry by law. It covers only injury to other parties, basically mandating that every driver protect others around them (but not themselves), just like in the United States. If a driver wants to protect him- or herself, he or she must purchase another policy, known as a Voluntary policy.

Thus:

- In the first two cases, the calendar to use in formatting `EffectiveDate` and `ExpirationDate` is dependant on the locale of the user only. For this, you merely need to set the `defaultCalendar` attribute on `<GWLocale>` (in `localization.xml`) to the correct value.
- In the last two cases, ClaimCenter displays these fields depending on additional conditions. For this, you need to provide additional configuration. See "Setting a Field to Conditionally Display the Japanese Imperial Calendar" on page 66.

See Also

- "Setting the Japanese Imperial Calendar as the Default for a Region" on page 65
- "Setting a Field to Always Display the Japanese Imperial Calendar" on page 65
- "Setting a Field to Conditionally Display the Japanese Imperial Calendar" on page 66

Sample JIC Presentation Handler

The following sample code illustrates how to create a Japanese Imperial Calendar (JIC) presentation handler. The code returns a JIC date for Japanese citizens. It also returns a JIC date if the loss location is in Japan.

```
package gw.datatype.presentation

uses gw.datatype.handler.IDatePresentationHandler
uses gw.lang.reflect.IPropertyInfo
uses gw.datatype.DisplayCalendar
uses pel.jp.util.PELUtil

class JapaneseImperialCalendarPresentationHandler implements IDatePresentationHandler {

  override function getDisplayFormat(p0 : Object, p1 : IPropertyInfo) : String {
    return null //## todo: Implement me
  }

  @Param("ctx","The object displaying the date property")
  @Param("prop","Details of the date property")
  override function getCalendar(ctx : Object, prop : IPropertyInfo) : DisplayCalendar {
    gw.api.util.Logger.logInfo("~~~ Running overridden getCalendar method")
    gw.api.util.Logger.logInfo("~~~ ctx is: " + ctx)
    gw.api.util.Logger.logInfo("~~~ prop is: " + prop.Name)

    //Only show Imperial birthdates for Japanese citizens
    if(ctx typeis Person and prop.Name == "DateOfBirth" and ctx.Nationality == Country.TC_JP) {
      gw.api.util.Logger.logInfo("~~~ Person.date of type = " + prop.Name)
      return JAPANESEIMPERIAL
    }
    //Show Imperial date if the loss location is Japan
    else if(ctx typeis Exposure and ctx.Claim.LossLocation.Country == Country.TC_JP) {
      gw.api.util.Logger.logInfo("~~~ Imperial date of type = " + prop.Name)
      return JAPANESEIMPERIAL
    }
    else {
      return GREGORIAN
    }
  }
}
```

part IV

National Formats and Data Configuration

chapter 10

Configuring Currencies

You can configure ClaimCenter with multiple currencies. With multiple currencies, you can specify monetary amounts in different currencies within a single instance of ClaimCenter.

This topic includes:

ClaimCenter Base Configuration Currencies

In the base configuration, Guidewire provides support for the following global currencies:

- `AUD` – Australian dollar
- `CAD` – Canadian dollar
- `EUR` – European Union euro
- `GBP` – British pound
- `JPY` – Japanese yen
- `RUB` – Russian ruble
- `USD` – U.S. dollar

You use the following files in working with a currency, depending on your installation:

Studio	Single currency rendering mode	Multicurrency rendering mode
`Currency` typelist	Contains currency code and similar information for the supported currency. • In a new installation, remove all currency typecodes except that of the default application currency. • In an installation that you are upgrading, retire all currency typecodes except that of the default application currency.	Contains information on all supported currencies (currency codes and similar information).
`currency.xml`	Do not use to define currency formats. Remove all currency formatting from this file. Instead, in single currency rendering mode, use file `localization.xml` to format currency information.	Studio contains multiple copies of `currency.xml`, one for each supported currency, for example: **configuration → config → currencies → aud → currency.xml**
`datatypes.xml`	Use to set the following for the application currency (the default application currency): • `precision` • `scale` • `appscale`	Use to set the following for the default currency only: • `precision` • `scale` • `appscale` Use individual `currency.xml` files to set the `storageScale` attribute on the `<CurrencyType>` element for each defined currency.
`localization.xml`	Contains currency formatting information for use with single currency rendering mode only. Studio contains multiple copies of this file, one for each locale, for example: **configuration → config → Localizations → en_US**	Do not use to define currency formats. Remove all currency formatting from `localization.xml`.

In addition, in working with currency and money amounts, you must set the following configuration parameters in `config.xml`:

- `DefaultApplicationCurrency` – Default application currency for currency and money amounts
- `DefaultRoundingMode` – Default rounding mode for currency and money amounts
- `MultiCurrencyDisplayMode` – Currency display mode in the application for selecting currencies

IMPORTANT If you integrate the core applications in Guidewire InsuranceSuite, you must set the values of `DefaultApplicationCurrency` and `MulticurrencyDisplayMode` the same in each application.

See also

- "Setting the Default Application Currency" on page 79
- "Choosing a Rounding Mode" on page 79
- "Setting the Currency Display Mode" on page 80

Working with Currency Typecodes

The typecodes in the `Currency` typelist determine which currencies you can use to enter monetary amounts in ClaimCenter. The base configuration of the `Currency` typelist defines the following currencies:

- U.S. Dollar
- Euro
- United Kingdom Pound

- Canadian Dollar
- Australian Dollar
- Russian Ruble
- Japanese Yen

For your configuration, add or retire currencies in the `Currency` typelist to suit your needs. The typelist must include at least one active currency.

Working with Currency Typecodes in the Currency Typelist

Depending on your installation type, you need to add, remove (delete), or retire existing typecodes from the `Currency` typelist.

Installation type	Action
Single currency install	Remove all currency typecodes except for the default application currency typecode.
Multiple currency install	Add or delete currency typecodes as representative for your installation.
Upgrading previous install	Retire all currency typecodes that you do not intend to use in the upgraded installation.

To add a currency to the Currency typelist

1. In Studio, open the `Currency` typelist using SHIFT+CTRL+N or by navigating the following path:

   configuration → **config** → **Extensions** → **Typelist**

2. In the toolbar, click the Add icon ✚.

   Studio adds a row for the typecode and selects it automatically.

3. Enter the following properties for the new currency.

code	Specify the lowercase version of the three-letter ISO 4217 currency code, for example, `nzd`.
name	Specify the uppercase version of the three-letter ISO 4217 currency code, for example, `NZD`.
desc	Specify the country or jurisdiction that issues the currency, followed by the name of the currency, for example, `New Zealand Dollar`.

Monetary Amounts in the Data Model and in Gosu

In the base configuration, Guidewire defines multiple data types to handle monetary amounts. You use the monetary data types to store and display monetary amounts within Guidewire ClaimCenter.

IMPORTANT Do not use the `<monetaryamount>` subelement in your data model. Do not use the corresponding `MonetaryAmount` Gosu or Java class. Use the `currencyamount` datatype instead.

Monetary Data Types

The `money` data types store and show monetary amounts in the system. Guidewire ClaimCenter assumes all monetary amounts in a `money` data type column are in the default application currency (as set by configuration parameter `DefaultApplicationCurrency`). The base configuration has the following `money` data types.

Money data type	Description
`money`	Permits positive, negative, and zero values.
`nonnegativemoney`	Does not permit negative values, However, zero values are acceptable.
`positivemoney`	Does not permit negative or zero values.

The `money` data types use the following specialized attributes, which you set in `datatypes.xml` for the default application currency:

- `precision`
- `scale`
- `appscale`

The settings that you make for these parameters affect all `money` columns.

Precision and Scale of Monetary Amounts

With `money` columns, precision controls the total number of digits and scale controls the number of digits to the right of the decimal point. For example, if `precision=5` and `scale=2`, then the maximum value for monetary amounts is 999.99 and the minimum value is -999.99.

Some factors to consider in choosing the scale value include globalization issues and specific business requirements. For example, to track monetary amounts down to 1/100 of the currency unit, then set `scale` to `4`. The default value for `scale` is `2`.

IMPORTANT The `MultiCurrencyDisplayMode` parameter setting is permanent. After you change the value of `MultiCurrencyDisplayMode` to `MULTIPLE` and then start the server, you cannot change the value back to `SINGLE` again. The `MultiCurrencyDisplayMode` parameter is in `config.xml`.

Guidewire recommends that you set the `scale` attribute to the maximum number of decimal position that you would possibly need in the future. Setting `scale` initially to a large value lets you later add currencies the number of decimal digits larger than your initial configuration.

You set `precision` and `scale` for the default application currency in the `<MoneyDataType>` element in file `datatypes.xml`, for example:

```
<MoneyDataType precision="18" scale="2" validator="Money"/>
```

The `precision` and `scale` attributes control how ClaimCenter displays monetary amounts in the user interface and how ClaimCenter stores monetary amounts in the database.

See also

- "The Data Types Configuration File" on page 258 in the *Configuration Guide*

Storage Scale in Multicurrency Installations

Installations that contain multiple currency definitions frequently need to set the number of decimal digits to store in the database differently for monetary amounts in different currencies. To set specific storage values for individual currencies, use the `storageScale` attribute on the `<CurrencyType>` element in file `currency.xml`. In installations that support the display of multiple currencies, there is a `currency.xml` file for every defined currency.

Attribute `storageScale` sets the number of decimal places to store in the database for money amounts in that currency. For example, in the base configuration:

File `currency.xml` for the Australian dollar defines the following `storageScale`:

```
<CurrencyType code="aud" desc="Australian Dollar" storageScale="2">
```

File `currency.xml` for the Japanese yen shows the following `storageScale`:

```
<CurrencyType code="jpy" desc="Japanese Yen" storageScale="0">
```

Application Scale

The `appscale` attribute is a rarely-used optional data type attribute. Guidewire provides the `appscale` attribute to facilitate upgrades of single currency configurations to configuration with multiple currencies. The value of `appscale` must be less than the value of `scale`.

Setting a value for the `appscale` attribute enables a single currency configuration to operate with a scale in the user interface that is appropriate to a particular currency. At the same time, monetary amounts stored in the database have a different and larger scale. The `appscale` value becomes the effective scale of `BigDecimal` values that ClaimCenter saves to and loads from monetary columns in the database.

For example, suppose that you implement a single currency configuration of ClaimCenter for the Japanese Yen, which typically has a scale of zero. You intend to convert to a multicurrency configuration in the future. In the conversion, you intend to add the U.S. Dollar, which typically has a scale of two.

In this example, you initially set `appscale=0` and `scale=2`. Thus in the interface, `BigDecimal` values for `money` and `currencyamount` columns have a scale of zero but are stored in the database with a scale of two. Later, during your conversion to multicurrency mode, you removed the `appscale` attribute from `datatypes.xml`. In its place, you use the `storageScale` attributes in the `currency.xml` configuration files for each currency to specify the scale of each currency.

Otherwise, you must recreate monetary columns during an upgrade to have a scale value of two. The Guidewire upgrade tool does not support changes in the scale of monetary columns.

The following rules and restrictions apply to the `money` data type.

- Set the value for `appscale` to the largest number of decimal positions that the currency you currently use requires. This value sets the scale for monetary amounts in the interface.
- Set the value for `scale` to the largest number of decimal positions that the currencies you plan to use in the future require. This value sets the scale for monetary amounts in the database.
- The value for `appscale` must be less than the value for `scale`.
- If you do not set a value for `appscale`, then ClaimCenter uses the value for `scale` in the user interface and in the database.

Guidewire does not use the `appscale` attribute in the base configuration. If you want to use this attribute, then you must add this attribute to the `<MoneyDataType>` element in `datatypes.xml`. Use the `appscale` attribute only in single currency mode. In multicurrency mode, use the `storageScale` attribute in the `currency.xml` file for each currency.

Currency Amount Data Types

Guidewire provides a number of `currencyamount` data types that extend the `money` data types to specify the currency of monetary amounts. The primary purpose of the `currencyamount` data types is for use in multicurrency configurations.

The following list describes the Guidewire base configuration of the `currencyamount` data types.

Currency amount data type	Description
`currencyamount`	Permits positive, zero, and negative.
`nonnegativecurrencyamount`	Permits only positive and zero values.
`positivecurrencyamount`	Permits only positive values.

In both Gosu and Java code, a `currencyamount` property returns a `CurrencyAmount` object, which associates a `BigDecimal` monetary amount with a particular `Currency` typekey. ClaimCenter persists the `BigDecimal` amount to the database. You can configure the `Currency` value for that particular column.

To specify the currency for amounts that are stored in a column, use the `<columnParam>` element on that column to set a `currencyProperty` value. The `currencyProperty` parameter points to a property on the entity that returns the `Currency` for the column. This property can be either of the following:

- A virtual property, which you typically define in an enhancement class
- A `Currency` column on the entity, which stores a `Currency` typekey value

If the property is a virtual property, it usually looks up the relevant `Currency` from a parent entity, such as a `Claim`.

For example, in ClaimCenter, if you want to store a monetary amount in the same currency as the claim, then you can use `ClaimCurrency` as the currency property.

```
<column name="SomeAmount" type="currencyamount" ...>
  <columnParam name="currencyProperty" value="ClaimCurrency"/>
</column>
```

See also “Multiple Currencies” on page 333 in the *Application Guide*.

In the base configuration, Guidewire provides a definition for the `ClaimCurrency` property on many entities, either in Java or through a Gosu enhancement class. If it does not exist, you can define it in an enhancement class on that particular entity type.

If you define a currency property, you must ensure that it does not throw a `NullPointerException` if the entity is not yet linked to its parent. Instead of throwing an exception, have the currency property return `null` if the entity that stores the currency value is not reachable. This is a relatively simple task in Gosu, because of the null-safety of entity path expressions. The following sample ClaimCenter code illustrates how to construct a null-safe `ClaimCurrency` property:

```
enhancement GWWorkStatusEnhancement : entity.WorkStatus {

    /**
     * Retrieves the currency of the claim associated with this WorkStatus object
     *
     * @return The associated Claim's currency, if any. Is null if the Claim is not
     *   currently reachable (for example, if the necessary entity connections have
     *   not yet been made).
     */
     property get ClaimCurrency() : Currency {
         return this.EmploymentData.ClaimCurrency
    }
    ....
}
```

It is possible to create a column that contains a multicurrency value that could potentially contain any `Currency`. In this case, you need to define another extension column on the entity to store the `Currency` value and reference that in the `currencyProperty` `<columnParam>` element. For example:

```
<typekey name="SomeCurrency" nullok="false" typelist="Currency" …/>
<column name="SomeAmount" type="currencyamount" ...>
  <columnParam name="currencyProperty" value="SomeCurrency"/>
</column>
```

It is not mandatory to provide the `currencyProperty` in a `<columnParam>`. If you do not, then ClaimCenter defaults to using the value that you set for `DefaultApplicationCurrency` in `config.xml`.

Exceptions Whenever Coercing BigDecimal Values to CurrencyAmount Values

If you attempt to coerce a value from `BigDecimal` to `CurrencyAmount`, the default application behavior creates a `CurrencyAmount` with a null `Currency` property. The base configuration sets the application server to strict currency mode, which typically generates exceptions for currency amounts with no designated currency. Guidewire recommends that you be aware of this issue and take steps to mitigate it.

The following sample Gosu code illustrates how coercing a `BigDecimal` value to a `CurrencyAmount` value can cause an exception.

```
var lineItem = new TransactionLineItem()
var bigDecimalAmount = new java.math.BigDecimal("1.23")

lineItem.setTransactionAmountAndUpdate(bigDecimalAmount) // Method takes a CurrencyAmount argument, so
```

```
Gosu coereces the BigDecimal value to a
CurrencyAmount with a null Currency.
```

See also

- "<column>" on page 185 in the *Configuration Guide*
- "<columnParam> Subelement" on page 188 in the *Configuration Guide*

Currency-related Configuration Parameters

The following list describes the configuration parameters that you must set in file `config.xml` that relate to currency.

Configuration parameter	Description	See...
`DefaultApplicationCurrency`	Default currency for the application	"Setting the Default Application Currency" on page 79
`DefaultRoundingMode`	Default rounding mode for money and currency amount calculations	"Choosing a Rounding Mode" on page 79
`MultiCurrencyDisplayMode`	Determines whether ClaimCenter displays currency selectors for monetary values	"Setting the Currency Display Mode" on page 80

IMPORTANT If you integrate the core applications in Guidewire InsuranceSuite, you must set the values of `DefaultApplicationCurrency` and `MulticurrencyDisplayMode` the same in each application.

See also

- "Globalization Parameters" on page 62 in the *Configuration Guide*

Setting the Default Application Currency

You must set configuration parameter `DefaultApplicationCurrency` to a typecode in the `Currency` typelist. You must set this configuration parameter even if you configure ClaimCenter with a single currency.

You set the default application currency in file `config.xml`, for example:

```
<param name="DefaultApplicationCurrency" value="usd"/>
```

The base ClaimCenter configuration sets the default application currency to the U.S. dollar by default.

WARNING The `DefaultApplicationCurrency` parameter setting is permanent. After you set the parameter and then start the server, you cannot change the value.

In most cases:

- ClaimCenter assumes that columns of type `money` are in the default application currency.
- ClaimCenter also assumes that `currencyamount` columns without a `currencyProperty <columnParam>` are in the default application currency.

Choosing a Rounding Mode

There are a number of factors to consider as you select a rounding mode, especially for Payments and Reserves. If you are creating a multicurrency transaction, ClaimCenter converts the `TransactionAmount` that you enter in the transaction currency to the claim currency. ClaimCenter then stores this amount as the `ClaimAmount`. The rounding mode that you stipulate and which ClaimCenter uses in this calculation can result in unexpected behavior at a later time as ClaimCenter makes Payments against the Available Reserves.

By default, ClaimCenter enforces a constraint that demands that the payments in the claim currency not exceed the reserves in the claim currency. If the selected rounding modes result in rounding upwards for particular payment amounts, and downwards for particular reserve amounts, then it is possible to violate this rule. This can occur even though the result does not exceed the Available Reserves if you consider the Transaction Amounts alone.

Therefore, Guidewire recommends that you make a careful selection of the rounding modes for the payments and the reserves. For example, you do not want ever to have the sum of the rounded claim-currency payments exceed the sum of the rounded claim-currency reserves. The default rounding modes for Payments and Reserves of `Down` and `Up`, respectively, achieve this invariant. Other combinations are possible as well.

Guidewire recommends that you choose `HALF_UP` or `HALF_EVEN` for both Payment and Reserve rounding modes in the following cases:

1. A downstream system, such as your General Ledger accounting system, expects a particular rounding mode to be used while determining the conversion.
2. You consider using `HALF_UP` or `HALF_EVEN` rounding to be more correct, and are comfortable with the possible side effect listed in the previous discussion.

See also

- "DefaultRoundingMode" on page 63 in the *Configuration Guide*
- "PaymentRoundingMode" on page 60 in the *Configuration Guide*
- "ReserveRoundingMode" on page 60 in the *Configuration Guide*

Setting the Currency Display Mode

You use configuration parameter `MultiCurrencyDisplayMode` in `config.xml` to set the currency display mode for your instance of ClaimCenter. You must set a currency display mode for ClaimCenter. Set the parameter to one of the following values:

- `SINGLE`
- `MULTIPLE`

The currency display mode determines how ClaimCenter stores monetary values and displays them in the interface.

IMPORTANT The `MultiCurrencyDisplayMode` parameter setting is permanent. After you change the value of `MultiCurrencyDisplayMode` to `MULTIPLE` and then start the server, you cannot change the value back to `SINGLE` again. The `MultiCurrencyDisplayMode` parameter is in `config.xml`.

The value that you set for MultiCurrencyDisplayMode influences the column type you use in your data model.

Currency display mode	Recommendations
SINGLE	If you set the currency mode to SINGLE, you can use the money data types for monetary amount columns. Use this value if you do not foresee a need for multicurrency operation in the future. With the parameter value set to SINGLE, ClaimCenter uses one currency only. So, there can be no ambiguity of the currency for monetary amounts that are stored in the database. However, there are advantages in using the currencyamount data types, even if you set the currency mode to SINGLE: • The currencyamount data types assure the accuracy of monetary amounts in the system because ClaimCenter stores monetary amounts in the database with the currency that the amounts represent. • Your Gosu code can pass around CurrencyAmount data types for monetary amounts with no ambiguity about the currency that the amounts represent. If you switch the currency mode to MULTIPLE at a later time, you avoid the necessary conversion of your Gosu code from money data types to currencyamount data types. Use the currencyamount data types if you set the currency mode to SINGLE and you foresee the remotest possibility of a need for multicurrency operation in the future.
MULTIPLE	If you set the currency mode to MULTIPLE, use only the currencyamount data types for monetary amount columns. With the parameter value set to MULTIPLE, ClaimCenter uses several currencies, so monetary amounts stored in the database carry the currencies that the amounts represent. If you set the currency mode to MULTPLE, your Gosu code must pass around only CurrencyAmount data types for monetary amounts. If you convert the currency mode from SINGLE to MULTIPLE, you can change the type on money extension columns to currencyamount without any upgrade cost. This change triggers a database upgrade, but the upgrade utility does not make changes to the database because the database types of the monetary columns do not change.

The value that you set for MultiCurrencyDisplayMode influences where you specify regional formats for monetary amounts.

Specifying Regional Formats for the Currency in Single Currency Display Mode

In single currency display mode, ClaimCenter assumes that all monetary amounts in the system are in the same currency. ClaimCenter uses the <CurrencyFormat> entries in each <GWLocale> in localization.xml to format the money amount, depending on the locale of each user. For example:

```
<Localization xmlns="http://guidewire.com/localization">
  <GWLocale code="en_US" name="English (US)" typecode="en_US">
    ...
    <CurrencyFormat negativePattern="($#)" positivePattern="$#" zeroValue="-"/>
  </GWLocale>
</Localization>
```

As all money values are in the default currency, you must ensure that the <CurrencyFormat> for each <GWLocale> specifies the money format for that one currency. It is possible to set slightly different formatting based on local custom, but all money formatting must be for the one default currency.

In single currency display mode, ClaimCenter assumes that all monetary amounts in the system are in the same currency, as specified by configuration parameter DefaultApplicationCurrency. ClaimCenter formats the display of monetary amounts using the CurrencyFormat entry in the GWLocale definition for the user's preferred regional formats.

As all monetary values are in the same currency, you must ensure that the `CurrencyFormat` for each `GWLocale` specifies the monetary format for that one currency. It is possible to set slightly different formats based on local custom, but all monetary formats must be for the single currency.

IMPORTANT Guidewire strongly recommends that you set configuration parameter `DefaultApplicationCurrency` to its correct value if you set `MulticurrencyDisplayMode` to `SINGLE`. This ensures the correctness of the data in the database if you upgrade to multicurrency at a later time.

Specifying Regional Formats for Different Currencies in Multiple Currency Display Mode

In multiple currency display mode, ClaimCenter obtains regional formats for monetary amounts from `currencies.xml`. For example:

```
<CurrencyType code="usd" desc="US Dollar" storageScale="2">
  <CurrencyFormat positivePattern="$#" negativePattern="($#)" zeroValue="-" />
</CurrencyType>
```

In `MULTIPLE` mode, ClaimCenter ignores any `CurrencyFormat` information in file `localization.xml`. However, even though unused, a tag for the default currency must be present in file `localization.xml`, even in `MULTIPLE` mode.

chapter 11

Configuring Geographic Data

This topic describes how to configure the typelists `Jurisdiction`, `Country`, and `State`, and configuration file `zone-config.xml` for the territorial data in your instance of ClaimCenter.

This topic includes:

- "Working with Country Typecodes" on page 83
- "Configuring Country Address Formats" on page 83
- "Setting the Default Application Country" on page 84
- "Configuring Jurisdiction Information" on page 85
- "Configuring Zone Information" on page 85

Working with Country Typecodes

In the base configuration, Guidewire provides a `Country` typelist (`Country.ttx`) that defines a standard set of countries. The country information includes the ISO country code and the English language country name. The ISO country code is the two-character uppercase ISO 3166 code for a country or region. To view a list of countries codes, refer to the following web site:

```
http://www.iso.org/iso/country_codes/iso_3166_code_lists /country_names_and_code_elements.htm
```

Individual country definitions can provide additional country information as well. For example, the entry for `VA` (Vatican City) indicates the currency in use is the euro (`eur`).

Configuring Country Address Formats

File `country.xml` defines settings that control the appearance of address information in ClaimCenter. Within Studio, there is a `country.xml` file for each defined country. ClaimCenter stores the `country.xml` files in the `geodata` folder, each in its own individual country folder. For example, the `country.xml` file for the address-related information in Australia resides in the following location in Studio:

configuration → config → geodata → AU

In the base configuration, Guidewire provides `country.xml` definition files for the following countries:

- `AU` – Australia
- `CA` – Canada
- `DE` – Germany
- `FR` – France
- `GB` – Great Britain
- `JP` – Japan
- `US` – United States

You use `country.xml` to set address-related configuration for a single country. In this file, you can set the following attributes on `<Country>`:

Attributes	Sets...
`PCFMode`	PCF mode for `GlobalAddressInputSet.pcf`. In the base configuration, Guidewire provides three modes for showing address information. The base configuration modes are: • `BigToSmall` – Used to format addresses for Japan • `PostCodeBeforeCity` – Used to format addresses for France and Germany • `default` – Used to format all other addresses It is possible for you to add new PCF modes and use them in file `country.xml`.
`postalCodeDisplayKey`	Name of the display key to use for the postal code label in ClaimCenter. For example, in the United States: `Web.AddressBook.AddressInputSet.ZIP`
`stateDisplayKey`	Name of the display key to use to designate a smaller geographical entity within a country. For example: • In the United States, this is State – `Web.AddressBook.AddressInputSet.State` • In Canada, this is Provinces – `Web.AddressBook.AddressInputSet.Province`
`visibleFields`	Comma-separated list of address-related fields that you want to be visible in ClaimCenter for this country, for example: • `Country` • `AddressLine1` • `AddressLine2` • `AddressLine3` • `City` • `State` • `PostalCode` Any address field that you designate as visible in this file must correspond to the constants defined in `AddressOwnerFieldId.gs`.

See “Address Modes in Page Configuration” on page 94 for more information on how these address configuration attributes work.

Setting the Default Application Country

You set the default application country in `config.xml`, in configuration parameter `DefaultCountryCode`. The country code must be a valid ISO country code that exists as a typecode in the `Country` typelist. See the following web site for a list of valid ISO country codes:

```
http://www.iso.org/iso/english_country_names_and_code_elements
```

See “Globalization Parameters” on page 62 in the *Configuration Guide* for more information on this configuration parameter.

Configuring Jurisdiction Information

Guidewire applications divide jurisdictions into several areas:

- National jurisdictions – Japan or France, for example
- State or province jurisdictions – United States and Canada, for example
- Other jurisdictions – Other local regulatory jurisdictions at a level below the country level, Berlin (Germany), for example

In the base configuration, Guidewire provides a `Jurisdiction` typelist that contains a number of pre-defined jurisdictions. Many ClaimCenter fields that seemly reference a state actually reference a jurisdiction instead, for example:

- `TaxLocation.State`
- `TaxLocationSearchCriteria.State`
- `TerritoryLookupCriteria.State`
- ...

Configuring Zone Information

Guidewire ClaimCenter uses `zone-config.xml` files to define one or more zones. A *zone* is a combination of a country and zero-to-many address elements such as a city, or state (United States), or province (Canada), for example. You can configure zones to apply to any area within a single country. In the United States, for example, you typically define zones by state, city, county, and ZIP (postal) code.

ClaimCenter uses zones for the following:

- Region and address auto-fill
- Business week and holiday definition by zone

You access `zone-config.xml` through Studio in the **Other Resources** folder. Within this file, you define the links between zones (the zone hierarchy) and how ClaimCenter uses those links to extract the value of zone from address data.

In the base configuration, ClaimCenter defines the zone hierarchy for the United States using the following elements:

```
<Zones countryCode="US">

  <Zone code="zip" fileColumn="1" regionMatchOrder="1" granularity="1" unique="true">
    <AddressZoneValue>Address.PostalCode.substring(0,5)</AddressZoneValue>
    <Links>
      <Link toZone="city"/>
    </Links>
  </Zone>

  <Zone code="state" fileColumn="2" orgZone="true" regionMatchOrder="3" granularity="4">
    <AddressZoneValue>Address.State.Code</AddressZoneValue>
    <Links>
      <Link toZone="zip" lookupOrder="1"/>
      <Link toZone="county"/>
      <Link toZone="city"/>
    </Links>
  </Zone>

  <Zone code="city" fileColumn="3" granularity="2">
    <ZoneCode>state + “:” + city</ZoneCode>
    <AddressZoneValue>Address.State.Code + “:” + Address.City</AddressZoneValue>
  </Zone>

  <Zone code="county" fileColumn="4" regionMatchOrder="2" granularity="3">
    <ZoneCode>state + “:” + county</ZoneCode>
    <AddressZoneValue>Address.State.Code + “:” + Address.County</AddressZoneValue>
    <Links>
      <Link toZone="zip" lookupOrder="1"/>
      <Link toZone="city" lookupOrder="2"/>
    </Links>
  </Zone>

</Zones>
```

Base Configuration Zone Hierarchies

In the base configuration, Guidewire defines zone hierarchies for the following geographical locations:

- `AU` – Australia
- `CA` – Canada
- `DE` – Germany
- `FR` – France
- `GB` – Great Britain
- `JP` – Japan
- `US` – Unites States of America

In the base configuration, Guidewire provides multiple `zone-config.xml` files, one for each supported zone. Guidewire stores each zone file in Studio in a separately labeled `geodata` folder:

configuration → **config** → **geodata** → **xx**

For example, the `zone-config.xml` file for the zone information in Australia resides in the following location in Studio:

configuration → **config** → **geodata** → **AU**

Working with Zone Configuration Files

A `zone-config.xml` files contains the following XML elements:

- `<Zones>`
- `<Zone>`
- `<ZoneCode>`
- `<Links>`
- `<AddressZoneValue>`

See also

- "Zone Import Command" on page 186 in the *System Administration Guide*.

<Zones>

This element defines the largest region, a country. The `zone-config.xml` files contains a single `<Zones>` element representing the zones in a specific country. Each `<Zones>` element contains one or more `<Zone>` elements.

The `<Zones>` element takes the following attributes:

Attribute	Description
`countryCode`	*Required.* Defines the country to which this zone configuration applies.
`dataFile`	For Guidewire internal use only. Ignore.

<Zone>

This element defines a zone type. The zone type must exist in the `ZoneType` typelist. The `Zone` element takes the following attributes, all optional except for the `code` attribute:

Attribute	Description
`code`	Sets the zone type. This must be a valid value defined in the `ZoneType` typelist. You also use this value as a symbol in `<ZoneCode>` expressions to represent the data extracted from the data import file based on the column specified in the `fileColumn` attribute.

`fileColumn`	*Optional.* Specifies the column in the import data file from which to extract the zone data. The numeric value of `fileColumn` indicates the ordered number of the column in the data file. For example, `fileColumn="4"` specifies the fourth column in the data file.
	
	Frequently (but not exclusively), a `<Zone>` element without a `fileColumn` attribute contains an expression that derives a value from the other zone values. For example, in the base configuration, Guidewire defines the following `fsa` zone in the Canadian `<Zones>` element:
	
	`<Zone code="fsa" regionMatchOrder="1" granularity="1">`
	`  <ZoneCode> postalcode.substring(0,3) </ZoneCode>`
	`  <AddressZoneValue> Address.PostalCode.substring(0,3) </AddressZoneValue>`
	`</Zone>`
	
	Notice that both the `<ZoneCode>` and `<AddressZoneValue>` elements extract data from the actual address data by parsing the data into substrings.
	
	Note:
	• You need to specify at least one `<Zone>` element with a `fileColumn` attribute. If you do not specify at least one `fileColumn` attribute, then ClaimCenter does not import data from the address zone data file.
	• You need to import address zone data upon first installing Guidewire ClaimCenter and then at infrequent intervals thereafter as you update zone definitions.

`granularity`	Sets the level of granularity for each defined zone. The smallest geographical region is 1. The next larger geographical region is 2, and so on. The sequence of numbers must be continuous. ClaimCenter uses this value with holidays and business weeks. For ClaimCenter Catastrophe administration, you must set the granularity for zones that you want to make available in the list of **Zone Types** on the **New Catastrophe** page.
`orgZone`	For Guidewire internal use only. Ignore.

`regionMatchOrder`	Controls the order in which ClaimCenter uses these zones in matching algorithms. ClaimCenter uses this attribute as it matches users to a zone for location- or proximity-based assignment. For example, in the base configuration, Guidewire defines the following `<Zone>` for `county`:
	
	`<Zone code="county"`
	`      fileColumn="4"`
	`      regionMatchOrder="2"`
	`      granularity="3">`
	`  ...`
	`</Zone>`
	
	Setting the `regionMatchOrder` to 2 means that ClaimCenter matches county data second, after another zone, while matching a user to a location.
	
	Notice also that the `county` value:
	• Appears in the fourth column of the data file.
	• Is third in `granularity`, one size less than a state (`granularity` 4) and one size more than a city (`granularity` 2).

`unique`	*Optional.* Specifies whether this zone data is unique. For example, in the United States, a `county` data value by itself does not guarantee uniqueness across all states. (There is a county with the name of *Union* in seventeen states and a county with the name of *Adams* in twelve states.) In situations such as this, use a `<ZoneCode>` element to construct a zone expression that uniquely identifies that zone data. For example, you can combine the county name with the state name to make a unique identifier. See “<ZoneCode>” on page 88 for additional information.

<ZoneCode>

It is possible for zone information to not be unique, meaning that the zone import data column contains two or more identical values. If this is the case, then you need to build an expression to uniquely identify the individual zone data using symbols.

For example, in the United States, it is possible for multiple states to have a city with the same name (Portland, OR and Portland, ME). Thus, to uniquely identify the city, you need to associate a particular state with the city as well. To do this, you create an expression that prepends the state import data value to the city import data value to obtain a unique city-state code. For example:

```
<ZoneCode>state + “:” + city</ZoneCode>
```

This expression instructs ClaimCenter to concatenate the State value with the County value with a semicolon delimiter. Only use values to construct the expression that are valid `<Zone>` code values (other than constants) that you defined within a `<Zones>` element for this country.

<Links>

This element defines one or more `<Link>` elements, each of which defines a connection (a link) between one zone type and another. For example, in the base configuration, Guidewire provides a `<Link>` element that defines a link between a county and a ZIP code in the United States. You can also use a link to define a lookup order to speed up searches.

The `<Link>` element contains the following attributes:

Attributes	Description
`lookupOrder`	*Optional.* Tells the application in what order to apply this value while looking up users by zone. This can increase performance somewhat. If you do not specify a `lookupOrder` value, ClaimCenter uses the order as it appears in the file.
`toZone`	*Required.* Defines a relationship to another zone (region) for ClaimCenter to use if the `<Zone> code` value is not available for lookup.

For example, the following code defines a link or relationship between one zone type (`county`) and several other zone types. ClaimCenter uses these zone types to look up an address if the address does not contain a `county` value.

```
<Zone code="county" fileColumn="4"  regionMatchOrder="2">
  ...
  <Links>
    <Link toZone="zip" lookupOrder="1"/>
    <Link toZone="city" lookupOrder="2"/>
  </Links>
</Zone>
```

This code has the following meaning:

- The first `<Link>` definition, `<Link toZone="zip" lookupOrder="1"/>`, creates a link from `county` to `zip`. If ClaimCenter cannot look up the address by its `county` value, then ClaimCenter attempts to look up the address first by `zip` value (`lookupOrder="1"`).
- The second `<Link>` definition instructs ClaimCenter to look up an address by its `city` value (`lookupOrder="2"`) if the `county` value is not available.

<AddressZoneValue>

This optional element uses an expression to define how to extract the zone code from an `Address` entity. Use entity dot notation, such as `Address.PostalCode`, to define a value for this element. These expressions can be subsets of an element or a concatenation of elements.

ClaimCenter extracts a value from the address data that uses this element and matches the value against zone data in the database. For example, in the base configuration, ClaimCenter defines a `<Zone>` element of type `postalcode` for Canada. It looks similar to the following:

```
<Zones countryCode="CA">
    <Zone code="postalcode"  fileColumn="1" unique="true">
        <AddressZoneValue> Address.PostalCode
    </AddressZoneValue>
</Zone>
```

Given this definition, ClaimCenter uses the value of the `PostalCode` field on the `Address` entity as the value of the `postalcode` zone.

Note: Guidewire recommends that you set this value even if there is a property defined on the `Address` entity that has the same name as the `Zone` name.

chapter 12

Configuring Address Information

Address data and formats in Guidewire ClaimCenter vary by country. For example, ClaimCenter shows a ZIP code or postal code field for an address dependent upon the country in which the address occurs.

This topic includes:

Understanding Global Addresses

Guidewire ClaimCenter displays address information in one of two ways:

- Text-entry fields
- Read-only text

The information that you see for an address depends on the country of the address and whether ClaimCenter displays the address as text-entry fields or read-only text.

The Modal Address PCF

In general, the country of an address determines which address fields in the database are visible in the ClaimCenter user interface for that address. The page configuration file `GlobalAddressInputSet` has modal versions that make different sets of address fields visible. The modal versions of `GlobalAddressInputSet` also control the order in which ClaimCenter displays address fields.

One paradigm for the modal versions of PCF `GlobalAddressInputSet` is a modal version for each country in which you want to permit addresses. However in practice, the addresses of different countries follow a small number of patterns in terms of components of an address and their order of presentation. Components of an address include the street name, the house number, the city, and country. Some countries have additional address components.

Aside For example, the country of an address is the component furthest from recipient of mail or visitors. Street address, including a building number, is the component nearest the recipient.

In the base configuration, Guidewire provides the following modal versions of PCF `GlobalAddressInputSet`.

Address PCF modal version	Address style	Used for…
`GlobalAddressInputSet.BigToSmall`	The address component furthest from the recipient comes first	• Japan
`GlobalAddressInputSet.default`	The address component closest to the recipient comes first, with postal codes following cities	• Australia • Canada • Great Britain • United States
`GlobalAddressInputSet.PostCodeBeforeCity`	The address component closest to the recipient comes first, with postal codes preceding cities	• France • Germany

See also

- For more information on modal PCF files, see “Working with Shared or Included Files” on page 295 in the *Configuration Guide*.

Addresses in Countries with States or Provinces

The country of an address controls the allowed values for the state or province field of the address. The state setting controls the value of Jurisdiction State, which you can change independently from the value for Address State. For countries that do not have states or provinces, ClaimCenter hides the state and province fields. For example, In Japan, ClaimCenter displays the Kanji fields. In Canada, ClaimCenter displays the Province field instead of the State field

See also

- “Configuring Jurisdiction Information” on page 85

Address Configuration Files

The following configuration files play a role in address configuration.

configuration → config → extensions → typelist	`State.ttx`	
	`StateAbreviations.ttx`	
	`Jurisdiction.ttx`	
	`Country.ttx`	
configuration → config → geodata	`address-config.xml`	
	`country.xml`	
	`zone-config.xml`	
	`countryCode``-locations.txt`	Mapping between postal codes and cit-ies for a country

Configuring Address Data and Field Order for a Country

You use country-specific `country.xml` files to configure the data and order that ClaimCenter uses to display addresses for specific countries. ClaimCenter stores `country.xml` files in country-specific folders in the `geodata` folder in Studio:

configuration → config → geodata

A country maps to an address mode through Gosu class `AddressCountrySettings`.

File `country.xml` defines the following address-related attributes:

Attribute	Description	See...
`PCFMode`	Order in which to display address fields	“Country Attribute for PCF Mode” on page 93
`postalCodeDisplayKey`	Label for the postal code field	“Country Attribute for Postal Code Display Key” on page 93
`stateDisplayKey`	Label for state or province field	“Country Attribute for State Display Key” on page 93
`visibleFields`	Which address fields to display	“Country Attribute for Visible Fields” on page 94

Country Attribute for PCF Mode

The attribute `PCFMode` determines which modal version of the address PCF ClaimCenter uses for specific countries. ClaimCenter uses the PCF mode to determine the following:

- The address fields to render for a specific country
- The order in which to render the address fields in a specific country

For example, country-specific `country.xml` files individually specify the PCF mode for the following countries:

France	`PCFMode="PostCodeBeforeCity"`
Japan	`PCFMode="BigToSmall"`

If a country-specific `country.xml` file does not have `PCFMode` defined, ClaimCenter uses the default modal version of address PCF. which is suitable generally for English-speaking countries.

See also

- “Address Modes in Page Configuration” on page 94

Country Attribute for Postal Code Display Key

The attribute `postalCodeDisplayKey` sets the display key to use as the label for the postal code of an address. For example, ClaimCenter uses the following display keys to label the postal code field for the following countries:

Japan	`postalCodeDisplayKey="Web.AddressBook.AddressInputSet.Postcode"`
United States	`postalCodeDisplayKey="Web.AddressBook.AddressInputSet.ZIP"`

If a country-specific `country.xml` file does not have `postalCodeDisplayKey` defined, ClaimCenter sets the value of the postal code display key value for the United States.

Country Attribute for State Display Key

The attribute `stateDisplayKey` sets the display key to use as the label for state or province of an address. For example, ClaimCenter uses the following display keys to label the state or province field for the following countries:

Japan	`stateDisplayKey="Web.AddressBook.AddressInputSet.Prefecture"`
United States	`stateDisplayKey="Web.AddressBook.AddressInputSet.State"`

If a country-specific `country.xml` file does not have `stateDisplayKey` defined, ClaimCenter sets the value of state display key to the value for the United States.

Country Attribute for Visible Fields

The attribute `visibleFields` defines the set of address fields that are visible for this country. For example:

France	`visibleFields="Country,AddressLine1,AddressLine2,AddressLine3,PostalCode,City, CEDEX,CEDEXBureau"`
Japan	`visibleFields="Country,PostalCode,State,City,CityKanji,AddressLine1,AddressLine1Kanji, AddressLine2,AddressLine2Kanji"`
United States	`visibleFields="Country,AddressLine1,AddressLine2,AddressLine3,City,State,PostalCode, County"`

If a country-specific `country.xml` file does not have `visibleFields` defined, ClaimCenter displays the same set of address fields as for the United States.

Address Modes in Page Configuration

Guidewire ClaimCenter uses a modal PCF for all addresses. Modal versions of the address PCF determine the order of fields in an addresses of specific countries.

In the base configuration, Guidewire provides the following modal versions of the address PCF.

Modal Address PCF	For addresses in …	For more information, see …
`GlobalAddressInputSet.BigToSmall`	• Japan	“Address Mode for Japan” on page 95
`GlobalAddressInputSet.PostCodeBeforeCity`	• France • Germany	“Address Mode for France and Germany” on page 96
`GlobalAddressInputSet.default`	• Australia • Canada • Great Britain • United States	“Address Mode for English-speaking Countries” on page 95

ClaimCenter stores the `GlobalAddressInputSet` PCF files in the following location in Guidewire Studio (starting from **Project**):

configuration → **config** → **Page Configuration** → **pcf** → **address**

Mapping Countries to Modes

A country maps to a mode through Gosu file `AddressCountrySettings`.

Controlling Field Properties

Each modal PCF uses Gosu class `AddressOwner` to control the following field properties:

- `available`
- `editable`
- `required`
- `visible`

Gosu Address Formatter

ClaimCenter uses Gosu class `AddressFormatter` to format the address display fields. It is possible to extend `AddressFormatter` to handle additional countries.

Address Mode for English-speaking Countries

In the base configuration, ClaimCenter defines the set of possible address fields for an address in an English-speaking country using the following PCF mode:

```
GlobalAddresInputSet.default
```

Address mode `default` displays read-only information for addresses in Australia, Canada, Great Britain, and the United States in the following format:

Read-only address fields
Address 1
Address 2
Address 3
City
County
State
ZIP Code

Read-only address fields	Example
Address 1	100 Main Street
Address 2	Suite 115
Address 3	(Not required)
City	Anchorage
County	Anchorage
State	Alaska
ZIP Code	99501

Address Mode for Japan

In the base configuration, ClaimCenter defines the set of possible address fields for a Japanese address using the following PCF mode:

```
GlobalAddresInputSet.BigToSmall
```

Address mode `BigToSmall` displays read-only information for addresses in Japan in the following format:

Read-only address fields
Postal code
Prefecture
City (Kanji)
Address 1 (Kanji)
Address 2 (Kanji)
Country

Read-only address fields	Example
Postal code	〒 100-0006
Prefecture	???
City (Kanji)	????

Read-only address fields	Example
Address 1 (Kanji)	有楽町 2-7-1
Address 2 (Kanji)	有楽町イトシア 12 階
Country	??

Address Mode for France and Germany

In the base configuration, ClaimCenter defines the set of possible address fields for a French or German address using the following PCF mode:

```
GlobalAddresInputSet.PostCodeBeforeCity
```

Address mode `PostCodeBeforecity` displays read-only information for addresses in France or Germany in the following format:

Read-only address fields
Address 1
Address 2
Address 3
Postal Code
City
CEDEX (checkbox)
CEDEX Bureau
Country

Read-only address fields	Example
Address 1	Musée d'Orsay
Address 2	
Address 3	
Postal Code	75007
City	Paris
CEDEX (checkbox)	(Not required)
CEDEX Bureau	(Not required)
Country	France

Automatic Address Completion and Fill-in

ClaimCenter provides automatic completion and fill-in of address information:

- **Automatic address completion** – As a user types characters in an address field, ClaimCenter displays a drop-down list choices, such as city names.
- **Automatic address fill-in** – After a user completes entering a value in an address field, ClaimCenter fills in other address fields with appropriate values. For example, after entering a postal code, ClaimCenter fills in the city and state/province fields automatically.

Configuring Automatic Address Completion

As a user enters in a value in one address field, ClaimCenter fills in values in other address fields.

For example, suppose a user enters a postal code of 99501 for a United States address. If configured, ClaimCenter sets the city, state, and county to that of Anchorage, Alaska, county of Anchorage.

To trigger the autofill functionality, the user must navigate away from the initial address field. It is also possible to trigger this functionality through the use of the autofill icon next to certain address fields.

You must configure automatic address completion as a separate step.

Configuring Automatic Address Fill-in

As a user types characters into an address field, ClaimCenter opens a drop-down list. The drop-down list displays possible completions of the entered characters based on the values in other address fields.

For example, suppose a user sets the State value in a United States address to CA (California). The user then moves to the City field and enters `Pa` as the start of a city name. If configured, ClaimCenter opens a drop-down list that shows the names of various cities in California that start with `Pa`, for example

- Palmdale
- Pasadena
- ...

The user can then select one of the items on the drop-down list for automatic entry into the address field.

You must configure automatic address fill-in as a separate step.

The Address Automatic Completion and Fill-in Plugin

Use the Address Automatic Completion plugin (`IAddressAutocompletePlugin`) to modify how automatic completion and fill-in operate in ClaimCenter. The base configuration of ClaimCenter provides the Java implementation class `DefaultAddressAutocompletePlugin`.

The Address Automatic Completion plugin interface defines the following functionality.

- The plugin provides a `createHandler` factory method for creating `AutocompleteHandler` objects. The default implementation creates an `AddressAutocompleteHandler`.
- You can add fields for automatic completion by extending `AddressAutofillable.eti`, `AddressAutofillDelegate.gs` and `AddressFillableExtension.gs`. Guidewire recommends that these three files always support exactly the same set of fields.

See also

- "Automatic Address Completion and Fill-in Plugin" on page 267 in the *Integration Guide*

chapter 13

Configuring Phone Information

Phone numbers are specific to each country. Guidewire ClaimCenter uses country information to determine the appropriate fields to show for user entry of the phone number. ClaimCenter also uses country information to correctly format a phone number in read-only mode.

This topic includes:

- “Working with Phone Configuration Files” on page 99
- “Setting Phone Configuration Parameters” on page 100
- “Phone Number PCF Widget” on page 100

Working with Phone Configuration Files

ClaimCenter stores phone-related configuration files in Studio in the following location (starting from **Project**):

configuration → **config** → **phone**

Important files in this directory include the following:

File	Description
`nanpa.properties`	Area codes as defined by the North American Numbering Plan Administration (NANPA).

Any change to an XML file in the `phone` directory requires that you regenerate the phone data in the `data` subdirectory. To regenerate phone data, run the following gwcc utility from the application `bin` directory:

```
gwcc regen-phone-metadata
```

For details on how to use the gwcc commands, see “Build Tools” on page 106 in the *Installation Guide*.

Setting Phone Configuration Parameters

You use configuration parameters to set phone-related information in ClaimCenter:

Configuration parameter	Sets...
`DefaultPhoneCountryCode`	The default ISO country code to use for phone data. The country code must be a valid ISO country code that exists as a typecode in the `PhoneCountryCode` typelist. See the following web site for a list of valid ISO country codes: `http://www.iso.org/iso/english_country_names_and_code_elements` The base application default phone country code is `US`.
`DefaultNANPACountryCode`	The default country code for region 1 phone numbers. If mapping file `nanpa.properties` does not contain the area code for this region, then ClaimCenter defaults to the area code value configured by this parameter. The base application default NANPA phone country code is `US`.
`AlwaysShowPhoneWidgetRegionCode`	Whether the phone number widget in ClaimCenter always shows a selector for phone region codes. The base application value for this parameter is `false`.

See "Globalization Parameters" on page 62 in the *Configuration Guide* for more information on this configuration parameter.

Phone Number PCF Widget

PCF widget `GlobalPhoneInputSet` provides a way to show phone-related fields in ClaimCenter. The phone-related fields that you see in ClaimCenter depends on country information that you enter and the screen mode (editable or read-only).

If in edit-mode, a ClaimCenter user has access to the following phone-related fields set in PCF widget `GlobalPhoneInputSet`:

- Country code
- National subscriber number
- Extension

If in read-only mode, a ClaimCenter user views previously entered phone-related information formatted in a specific way depending on the phone country code.

You initialize the `GlobalPhoneInputSet` by providing the `InputSetRef` with a `PhoneOwner`. You initialize a `PhoneOwner` by providing a `PhoneFields` object.

Phone Numbers in Edit Mode

International phone numbers begin with a country code, according to the following format.

```
+phoneCountryCode phoneNumber extensionNumber
```

It is also possible to have an extension to the phone number as well.

For ease of entering phone information, it is possible to configure the `GlobalPhoneInputSet` widget to show a list of **Region Code** values in ClaimCenter from which the user can chose. For ClaimCenter to show the **Region Code** drop-down, all of the following must be true:

- Configuration parameter `AlwaysShowPhoneWidgetRegionCode` in `config.xml` must be set to `true`.
- The user must enter a plus sign into the phone number field.

If the user enters a plus sign (`+`) in entering a phone number, the `GlobalPhoneInputSet` widget issues a post-on-change event to expose a **Region Code** drop-down list. Application logic maps the region code to a country code (in file `CountryCodesToRegionCodes.xml`) and identifies the corresponding country. ClaimCenter formats the phone number based on the user's default phone country and the region of the phone number, for example:

User country	Phone region	ClaimCenter formats number for...
`US`	`US`	Domestic United States
`US`	`CN`	International Chinese
`CN`	`CN`	Domestic Chinese
`CN`	`US`	International United States

If a user enters a number phone number without first entering a country code, then ClaimCenter invokes only the format action on a targeted post-on-change event. Also, ClaimCenter invokes only the format action if the country code is the same as the user's selected default or default configured for the server.

See also

- "AlwaysShowPhoneWidgetRegionCode" on page 65 in the *Configuration Guide*

Phone Numbers in the Read-only Mode

The read-only phone number field of the `GlobalPhoneInputSet` widget formats phone numbers based on one of the following:

- The default phone country code selected by the user
- The default phone country code configured for the server

Users select their default phone country codes through the standard preferences screen in ClaimCenter. A user's selection for default phone country code overrides the default of the server.

chapter 14

Configuring National Field Validation

Field validation in ClaimCenter generally relies on regular expressions and input masks to validate data that users enter in specific fields. Field validators define specific regular expressions and input masks. Sometimes, field validation varies by country. For example, many countries issue taxpayer IDs, but the validation rules for taxpayer IDs vary by country.

This topic includes:

- "Understanding National Field Validation" on page 103
- "Localizing Field Validators for National Field Validation" on page 104

See also

- "Field Validation" on page 249 in the *Configuration Guide*

Understanding National Field Validation

Field validators provide basic validation for data that users enter in specific fields.

- Field validators look only at the value in a single field.
- Field validators do not enforce the uniqueness of values in that field.
- Field validators generally ignore relationships between values in that field and values in other fields.

A field validator typically defines a *regular expression*, which is a pattern of characters and special symbols that a value entered in a text field must match to be valid. Optionally, field validators can define an *input mask*, which provides a visual indication to the user of the expected format for values to enter in the field.

You can configure national field validation for fields of data type `LocalizedString` only. In addition, any entity definition that contains localized string fields must have an additional field to store a country code associated with each entity instance. ClaimCenter applies national field validation based on the value of the country code associated with specific entity instances.

You configure field validation by using `fieldvalidtors.xml` files in the **fieldvalidators** package. You define global field validators once in the `fieldvalidtors.xml` file located in the root of the **fieldvalidators** package. You define national field validators in `fieldvalidtors.xml` files located in country-specific packages within the **fieldvalida-tors** package.

See also

- "Data Types" on page 255 in the *Configuration Guide*

Localizing Field Validators for National Field Validation

You define national field validators in `fieldvalidtors.xml` files located in country-specific packages within the **fieldvalidators** package. Country-specific package names must match typecodes from the `Country` typelist.

To define national field validators for a specific country

1. In Guidewire Studio, navigate to **configuration** → **config** → **fieldvalidators.**
2. Right-click **fieldvalidators**, and then select **New** → **package** from the context menu.
3. Enter the typecode from the `Country` typelist for the country, and then click **OK**.
4. Copy the `fieldvalidators.xml` file from the root of the **fieldvalidators** package to the new country-specific package.
5. Modify the copy of `fieldvalidators.xml` that you just made to define national field validators for the country.