

# Guidewire PolicyCenter®

## PolicyCenter Globalization Guide

RELEASE 8.0.2

Copyright © 2001-2014 Guidewire Software, Inc. All rights reserved.

Guidewire, Guidewire Software, Guidewire ClaimCenter, Guidewire PolicyCenter, Guidewire BillingCenter, Guidewire Reinsurance Management, Guidewire ContactManager, Guidewire Vendor Data Management, Guidewire Client Data Management, Guidewire Rating Management, Guidewire InsuranceSuite, Guidewire ContactCenter, Guidewire Studio, Guidewire Product Designer, Guidewire Live, Guidewire DataHub, Guidewire InfoCenter, Guidewire Standard Reporting, Guidewire ExampleCenter, Guidewire Account Manager Portal, Guidewire Claim Portal, Guidewire Policyholder Portal, ClaimCenter, BillingCenter, PolicyCenter, InsuranceSuite, Gosu, Deliver Insurance Your Way, and the Guidewire logo are trademarks, service marks, or registered trademarks of Guidewire Software, Inc. in the United States and/or other countries.

All other trademarks are the property of their respective owners.

**This material is confidential and proprietary to Guidewire and subject to the confidentiality terms in the applicable license agreement and/or separate nondisclosure agreement.**

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

Product Name: Guidewire PolicyCenter

Product Release: 8.0.2

Document Name: *PolicyCenter Globalization Guide*

Document Revision: 20-May-2014

# Contents

<b>About PolicyCenter Documentation</b> .....	<b>7</b>
Conventions in This Document .....	8
Support .....	8

## Part I

### Introduction

<b>1 Understanding Globalization</b> .....	<b>11</b>
Dimensions of Globalization .....	11
Shortcomings of the Two Traditional Globalization Dimensions .....	12
Globalization Dimensions in Guidewire Applications .....	12
Selecting Language and Regional Formats in PolicyCenter .....	13
Configuration Files Used for Globalization .....	15
Globalization Configuration Parameters in config.xml .....	17

## Part II

### Language Configuration

<b>2 Working with Languages</b> .....	<b>21</b>
About Display Languages. ....	21
About Language Hierarchies. ....	22
Installing Display Languages .....	23
About the Language Pack Installer .....	24
Installing a Language Pack by Using the Language Pack Installer .....	24
Installed Language Pack Files in the File System. ....	24
Installed Language Pack Files in Studio. ....	25
Activity Logging by the Language Pack Installer. ....	27
Upgrading Display Languages .....	28
Setting the Default Display Language .....	29
Selecting a Personal Language Preference .....	29
<b>3 Localized Printing</b> .....	<b>31</b>
Printing Specialized Character Sets and Fonts .....	31
Localized Printing in a Windows Environment .....	32
Step 1: Download and Install Apache FOP on Windows .....	32
Step 2: Configure TTFReader. ....	33
Step 3: Generate FOP Font Metrics Files .....	33
Step 4: Register the Fonts with Apache FOP .....	33
Step 5: Register Your Apache FOP Configuration and Font Family with PolicyCenter .....	34
Step 6: Test Your Configuration. ....	34
Localized Printing in a Linux Environment .....	34
Before Starting .....	34
Step 1: Download and Install the Required Fonts. ....	35
Step 2: Download and Install Apache FOP on Linux. ....	35
Step 3: Configure the Font .....	35
Step 4: Register the Font with Apache FOP. ....	35
Step 5: Register Your Apache FOP Configuration and Font Family with PolicyCenter .....	36
Step 6: Test Your Configuration. ....	36

<b>4</b>	<b>Localizing PolicyCenter String Resources</b>	<b>37</b>
	Understanding String Resources	38
	Display Keys	38
	Typecodes	38
	Workflow Step Names	39
	Exporting and Importing String Resources	39
	Exporting Localized String Resources with the Command Line Tool	39
	Importing Localized String Resources with the Command Line Tool	40
	Localizing Display Keys	41
	PolicyCenter and the Master List of Display Keys	41
	Localizing Display Keys by Using the Display Key Editor	42
	Identifying Missing Display Keys	42
	Working with Display Keys for Later Translation	43
	Localizing Typecodes	43
	Using the gwpc Export and Import Commands	43
	Using the Typelist Localization Editor	43
	Editing the typelist.properties file	44
	Setting Localized Sort Orders for Localized Typecodes	44
	Accessing Localized Typekeys from Gosu	45
	Localizing Gosu Error Messages	45
	Localizing Product Model String Resources	46
	Translating Product Model Strings in Product Designer	46
	Localizing Coverage Term Options	46
<b>5</b>	<b>Localizing PolicyCenter with Studio</b>	<b>49</b>
	Viewing Unicode Characters in Studio	49
	Localizing Rule Set Names and Descriptions	49
	Setting the IME Mode for Field Inputs	50
	IME and Text Entry	50
	Setting a Language for a Block of Gosu Code	51
<b>6</b>	<b>Localizing Administration Data</b>	<b>53</b>
	Understanding Administration Data	53
	Localized Columns in Entities	53
	Localization Attributes	54
	Localization Element Example	54
	Localization Tables in the Database	55
	System Table Localization	55
	The Product Designer System Table Editor	56
	What You Can Localize	56
<b>7</b>	<b>Localizing Guidewire Workflow</b>	<b>57</b>
	Localizing PolicyCenter Workflow	57
	Localizing Workflow Step Names	58
	Creating a Locale-Specific Workflow SubFlow	59
	Localizing Gosu Code in a Workflow Step	59
<b>8</b>	<b>Localizing Templates</b>	<b>61</b>
	About Templates	61
	Creating Localized Documents, Emails, and Notes	62
	Step 1: Create Locale-Specific Folders	62
	Step 2: Copy Template Content Files	63
	Step 3: Localize Template Descriptor Files	63
	Step 4: Localize Template Files	65
	Step 5: Localize Documents, Emails, and Notes in PolicyCenter	65

Document Localization Support . . . . .	66
IDocumentTemplateDescriptor Interface Methods . . . . .	67
IDocumentTemplateSource Plugin Interface Methods . . . . .	67
IDocumentTemplateSerializer Plugin Interface Methods . . . . .	67

## Part III

### Regional Format Configuration

<b>9 Working with Regional Formats . . . . .</b>	<b>71</b>
Configuring Regional Formats . . . . .	71
About the International Components for Unicode (ICU) Library . . . . .	71
Configuring Locale Codes for Default Application Locale and the ICU Library . . . . .	72
Configuring a localization.xml file . . . . .	72
Setting the Default Application Locale for Regional Formats . . . . .	75
Setting Regional Formats for a Block of Gosu Code . . . . .	75
<b>10 Working with the Japanese Imperial Calendar . . . . .</b>	<b>77</b>
The Japanese Imperial Calendar Date Widget . . . . .	77
Configuring Japanese Dates . . . . .	78
Setting the Japanese Imperial Calendar as the Default for a Region . . . . .	79
Setting a Field to Always Display the Japanese Imperial Calendar . . . . .	80
Setting a Field to Conditionally Display the Japanese Imperial Calendar . . . . .	80
Working with Japanese Imperial Dates in Gosu . . . . .	82

## Part IV

### National Formats and Data Configuration

<b>11 Configuring Currencies . . . . .</b>	<b>87</b>
PolicyCenter Base Configuration Currencies . . . . .	87
Working with Currency Typecodes . . . . .	88
Monetary Amounts in the Data Model and in Gosu . . . . .	89
Monetary Amount Data Type . . . . .	90
Currency-related Configuration Parameters . . . . .	92
Setting the Default Application Currency . . . . .	92
Choosing a Rounding Mode . . . . .	92
Setting the Currency Display Mode . . . . .	93
<b>12 Configuring Geographic Data . . . . .</b>	<b>95</b>
Working with Country Typecodes . . . . .	95
Configuring Country Address Formats . . . . .	95
Setting the Default Application Country . . . . .	97
Configuring Jurisdiction Information . . . . .	97
Configuring Zone Information . . . . .	97
Location of Zone Configuration Files . . . . .	97
Working with Zone Configuration Files . . . . .	98
<b>13 Configuring Address Information . . . . .</b>	<b>103</b>
Addresses in PolicyCenter . . . . .	103
Understanding Global Addresses . . . . .	104
Country XML Files . . . . .	105
Modal Address PCF Files . . . . .	105
AddressFormatter Class . . . . .	105
Addresses and States or Jurisdictions . . . . .	107
Address Configuration Files . . . . .	107

Configuring Address Data and Field Order for a Country . . . . .	107
Configuring the Country XML File . . . . .	108
Additional Country and Address Configurations . . . . .	109
Address Modes in Page Configuration . . . . .	110
Address Owners . . . . .	111
AddressOwnerFieldId Class . . . . .	111
Automatic Address Completion and Fill-in . . . . .	112
Configuring Automatic Address Completion . . . . .	112
Configuring Automatic Address Fill-in . . . . .	112
The Address Automatic Completion and Fill-in Plugin . . . . .	112
<b>14 Configuring Phone Information . . . . .</b>	<b>113</b>
Working with Phone Configuration Files . . . . .	113
Setting Phone Configuration Parameters . . . . .	114
Phone Number PCF Widget . . . . .	114
Phone Numbers in Edit Mode . . . . .	114
Phone Numbers in Read-only Mode . . . . .	115
<b>15 Linguistic Search and Sort. . . . .</b>	<b>117</b>
Linguistic Search and Sort of Character Data . . . . .	117
Effect of Character Data Storage Type on Searching and Sorting . . . . .	118
Character Data in the Database . . . . .	118
Character Data in Memory . . . . .	118
Searching and Sorting in Configured Languages . . . . .	118
Configuring Search in the PolicyCenter Database . . . . .	119
Searching and the Oracle Database . . . . .	119
Searching and the SQL Server Database . . . . .	122
Configuring Sort in the PolicyCenter Database . . . . .	122
Configuring Database Sort in language.xml . . . . .	123
Configuring Database Sort in collations.xml . . . . .	123
<b>16 Configuring National Field Validation . . . . .</b>	<b>125</b>
Understanding National Field Validation . . . . .	125
Localizing Field Validators for National Field Validation . . . . .	126

# About PolicyCenter Documentation

The following table lists the documents in PolicyCenter documentation.

Document	Purpose
<i>InsuranceSuite Guide</i>	If you are new to Guidewire InsuranceSuite applications, read the <i>InsuranceSuite Guide</i> for information on the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with Guidewire applications.
<i>Application Guide</i>	If you are new to PolicyCenter or want to understand a feature, read the <i>Application Guide</i> . This guide describes features from a business perspective and provides links to other books as needed. The intended readers are everyone who works with PolicyCenter.
<i>Upgrade Guide</i>	Describes how to upgrade PolicyCenter from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing PolicyCenter application extensions and integrations.
<i>New and Changed Guide</i>	Describes new features and changes from prior PolicyCenter versions. Intended readers are business users and system administrators who want an overview of new features and changes to features. Consult the “Release Notes Archive” part of this document for changes in prior maintenance releases.
<i>Installation Guide</i>	Describes how to install PolicyCenter. The intended readers are everyone who installs the application for development or for production.
<i>System Administration Guide</i>	Describes how to manage a PolicyCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
<i>Configuration Guide</i>	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files. The intended readers are all IT staff and configuration engineers.
<i>Globalization Guide</i>	Describes how to configure PolicyCenter for a global environment. Covers globalization topics such as 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.
<i>Rules Guide</i>	Describes business rule methodology and the rule sets in PolicyCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
<i>Contact Management Guide</i>	Describes how to configure Guidewire InsuranceSuite applications to integrate with ContactManager and how to manage client and vendor contacts in a single system of record. The intended readers are PolicyCenter implementation engineers and ContactManager administrators.
<i>Best Practices Guide</i>	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
<i>Integration Guide</i>	Describes the integration architecture, concepts, and procedures for integrating PolicyCenter 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.
<i>Gosu Reference Guide</i>	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
<i>Glossary</i>	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.

Document	Purpose
<i>Product Model Guide</i>	Describes the PolicyCenter product model. The intended readers are business analysts and implementation engineers who use PolicyCenter or Product Designer. To customize the product model, see the <i>Product Designer Guide</i> .
<i>Product Designer Guide</i>	Describes how to use Product Designer to configure lines of business. The intended readers are business analysts and implementation engineers who customize the product model and design new lines of business.

## Conventions in This Document

Text style	Meaning	Examples
<i>italic</i>	Emphasis, special terminology, or a book title.	A <i>destination</i> sends messages to an external system.
<b>bold</b>	Strong emphasis within standard text or table text.	You <b>must</b> define this property.
<b>narrow bold</b>	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Next, click <b>Submit</b> .
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.
<i>monospaced italic</i>	Parameter names or other variable placeholder text within URLs or other code snippets.	Use <code>getName(<i>first</i>, <i>last</i>)</code> . <code>http://<i>SERVERNAME</i>/a.html</code> .

## 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](mailto:support@guidewire.com)
- By phone – +1-650-356-4955





part I

# Introduction



# Understanding Globalization

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

This topic includes:

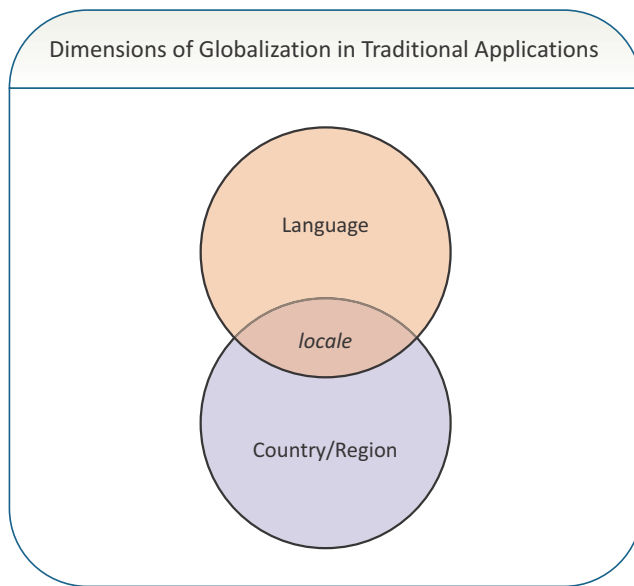
- “Dimensions of Globalization” on page 11
- “Selecting Language and Regional Formats in PolicyCenter” on page 13
- “Configuration Files Used for Globalization” on page 15

## Dimensions of Globalization

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

- **Language** – Writing system and words to use for text in the user interface
- **Country/region** – Formatting of 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 Java 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 Java 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 or 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

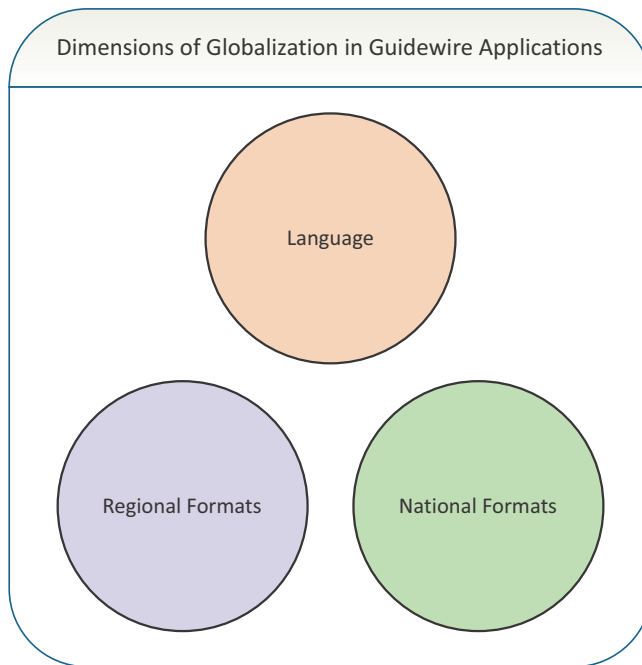
Furthermore, traditional applications enable users to select only pre-defined locales. For example, users typically cannot select French as the language, and, at the same time, select formats for dates, times, numbers, and monetary amounts used by convention in Great Britain.

## Globalization Dimensions in Guidewire Applications

Guidewire applications overcome the shortcomings of the traditional model by providing three dimensions for operating in a global environment. These three dimensions are independent:

- **Language** – Writing system and words to use for text in the user interface, as well as for linguistic searching and sorting behavior.
- **Regional formats** – Formatting of 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, but for which formats vary by regional convention.

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




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

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

## Selecting Language and Regional Formats in PolicyCenter

In Guidewire PolicyCenter, each user can set the following:

- The language that PolicyCenter uses to display labels and drop-down menu choices
- The regional formats that PolicyCenter 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 using the Options  menu at the top, right-hand side of the PolicyCenter screen. On that menu, click **International**, and then select one of the following:

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

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

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

PolicyCenter indicates the current selections for **Language** and **Regional Formats** by placing a check mark to the left of each selected option.

### Options for Language

In the base configuration, Guidewire has a single display language, English. To view another language in PolicyCenter, you must install a language pack and configure PolicyCenter for that language. If your installation has more than one language, 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, PolicyCenter uses the default language. The configuration parameter `DefaultApplicationLanguage` specifies the default language. In the base configuration, the default language is `en_US`, U.S. English.

### Options for Regional Formats

If your installation contains more than one configured region, you can select a regional format for that locale from the **Regional Formats** submenu. At the time you configure a region, 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** menu. The base configuration defines the following locale types:

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

Unless you select a regional format from the **Regional Formats** menu, PolicyCenter uses the regional formats of the default region. The configuration parameter `DefaultApplicationLocale` specifies the default region. In the base configuration, the default region is `en_US`, United States (English). If you select your preference for region from the **Regional Formats** menu, you can later use the default region again only by selecting it from the **Regional Formats** menu.

### See also

- “About Display Languages” on page 21
- “Working with Regional Formats” on page 71

## Configuration Files Used for Globalization

You use Guidewire Studio to edit the configuration files for globalization. The following list describes the configuration files and how to navigate to them in the **Project** window.

Location in Project window	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
	StateAbbreviations.ttx	
	Jurisdiction.ttx	
	Country.ttx	
configuration → config → Metadata → Typelist	LocaleType.tti	List of defined locales
configuration → config	config.xml	Configuration parameters related to localization. The localization-related configuration parameters include, among others: <ul style="list-style-type: none"> <li>• DefaultApplicationLocale</li> <li>• DefaultApplicationLanguage</li> <li>• DefaultCountryCode</li> <li>• DefaultPhoneCountryCode</li> </ul> Each Guidewire application instance contains a single copy of config.xml.
	collations.xml	Specialized rules for database sort and search for various languages and database types.

Location in Project window	Configuration file	Description
configuration → config → Localizations → <i>LocalizationFolder</i>	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. Each region must have a separate display.properties file. It is the presence of multiple display.properties files that alerts PolicyCenter to the fact that multiple locales exist.  The display.properties file (as with all property files) is a standard Java property file with the following format:  display_key_name = value.
	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	Contains Gosu error messages. Studio displays these messages if it encounters a Gosu error condition. You can translate these error messages into the languages of your choice.
	language.xml	Collation configuration for a specific language.
	productmodel.display.properties	Display keys for the fields used in the PolicyCenter product model. The text strings in this file are in the language specified by the name of the containing folder.
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	Format information for fields such as currencies, phone numbers, and ID fields, and other fields that need validation and input masks.
configuration → config → fieldvalidators → <i>LocalizationFolder</i>	fieldvalidators.xml	Field validator definition overrides by country.
configuration → config → geodata	LocaleCode-locations.txt	Mapping between postal codes and cities for a country.



Location in Project window	Configuration file	Description
configuration → config → geodata → <i>LocalizationFolder</i>	address-config.xml	Address configuration by country.
	country.xml	
	zone-config.xml	Zone information for a specific region. Zones are address components used for address autofill and region creation.
configuration → config → phone	nanpa.properties	Area codes as defined by the North American Numbering Plan Administration (NANPA). These area codes apply to North American countries other than the United States.
	PhoneNumberMetaData.xml	Area codes and validation rules for international phone numbers. Do not made changes to this file. See the comments at the beginning of the file for more information.
	PhoneNumberAlternateFormats.xml	Additional area codes and validation rules for international phone numbers. Do not made changes to this file. See the comments at the beginning of the file for more information.

## 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.

### Configuration Parameters for General Globalization Features

The following parameters in `config.xml` relate to the general globalization features of PolicyCenter. For more information about these parameters, see “Globalization Parameters” on page 56 in the *Configuration Guide*

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



# Language Configuration



# Working with Languages

PolicyCenter enables you to configure support for multiple display languages. Display languages specify the writing system and words to use for text in the user interface, as well as linguistic searching and sorting behavior. Generally, you configure PolicyCenter for display languages by installing language packs from Guidewire.

This topic includes:

- “About Display Languages” on page 21
- “Installing Display Languages” on page 23
- “Upgrading Display Languages” on page 28
- “Setting the Default Display Language” on page 29
- “Selecting a Personal Language Preference” on page 29

## About Display Languages

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

- **Install additional display languages** – Install additional language packs from Guidewire by using the language pack installer. Language packs 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 installed languages as the default display language for PolicyCenter. Users see the application in the default language at the time that they log in. A user can select any installed language as their preferred display language.

### Important:

PolicyCenter enables you to configure display languages of your choice manually. However, Guidewire does not provide customer support for either of the following configurations:

- Configuring PolicyCenter with a language for which Guidewire does not provide a language pack.

- Configuring PolicyCenter with a language for which Guidewire provides a language without installing the Guidewire language pack for that language.

**Note:** PolicyCenter provides features that might appear to allow you to manually configure display languages of your choice. However, those features require that you first install a Guidewire language pack, and they support localizing your extensions to PolicyCenter only for these installed languages.

**See also**

- “Installing Display Languages” on page 23
- “Setting the Default Display Language” on page 29

## About Language Hierarchies

You can configure PolicyCenter 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, you could set up 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** folder. You create a localization subfolder and name it by using a lower-case ISO language code that has only two letters. For example, in the Studio Project window, 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** folder to configure the root Spanish language. In the **es** folder, provide values for all the PolicyCenter display keys and other string resources localized to your version of the root Spanish language. The values that you 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** folder with localization folders that you name by using Java locale codes. A Java locale code is a language-country pair. It specifies the language by using a lower-case, two-letter ISO language code and it specifies the country by using an upper-case two-letter ISO country code.

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 and an **es\_MX** localization folder for Mexican Spanish. For example, viewed in the Studio Project window, the localization folders for the variants of English spoken in Spain and Mexico would be the following:

- **configuration** → **config** → **Localizations** → **es\_ES**
- **configuration** → **config** → **Localizations** → **es\_MX**

The display keys and typecodes defined in these folders override settings in the **es** folder, or add to them, if the equivalent language is selected.

The **es\_ES** localization folder would most likely hold very few display keys or other string resources. There are few variations between Spanish spoken in Spain and the **es** localization folder for root Spanish. In contrast, the **es\_MX** localization folder would most likely hold 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 British English requires more than just using **en** as the base folder because the English locale in the base configuration is **en\_US**. In this case, Guidewire recommends that you do the following:

1. Create an **en** folder and copy the **en\_US** property files into that folder.
2. Create an **en\_GB** folder, with the localized property files that contain only the specific keys and values for use in Great Britain.
3. Create additional folders for any other variants of English you want to provide. Add the localized properties files that contain only the specific keys and values for use in those countries.

For example, viewed in the Studio **Project** window, the localization folders for the variants of English spoken in Great Britain, Australia, and New Zealand would be the following:

- **configuration** → **config** → **Localizations** → **en\_GB**
- **configuration** → **config** → **Localizations** → **en\_AU**
- **configuration** → **config** → **Localizations** → **en\_NZ**

### See also

- “Installing Display Languages” on page 23
- “Upgrading Display Languages” on page 28
- “Activity Logging by the Language Pack Installer” on page 27

## Installing Display Languages

To use a display language other than U.S. English in PolicyCenter, you must install a language pack from Guidewire. To install a Guidewire language pack, you must use the language pack installer that Guidewire provides with PolicyCenter. See “About the Language Pack Installer” on page 24.

The language pack installer performs the following actions when it installs a new language pack:

- Verifies the integrity of the files in the language pack 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 pack was properly installed in your configuration of PolicyCenter.

After you install a language pack, the installation is permanent. Language packs cannot be removed.

This topic includes:

- “About the Language Pack Installer” on page 24
- “Installing a Language Pack by Using the Language Pack Installer” on page 24
- “Installed Language Pack Files in the File System” on page 24
- “Installed Language Pack Files in Studio” on page 25
- “Activity Logging by the Language Pack Installer” on page 27

### See also

- “Setting the Default Display Language” on page 29
- “Selecting a Personal Language Preference” on page 29
- “Upgrading Display Languages” on page 28

## About the Language Pack Installer

The language pack installer is a command line tool that enables you to install or upgrade installed display languages.

**Note:** This installer is also described in the release notes file for each Guidewire language pack.

The syntax for the command is:

```
gwpc 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 pack. Be certain to include the .zip extension. The file must have been saved in the root of the PolicyCenter home directory.
Dinstall.type	The type of operation for the language pack installer to perform: <ul style="list-style-type: none"> <li>install – Install a language pack for a language that is not currently installed.</li> <li>upgrade – Upgrade an installed language with a newer version of the language pack.</li> </ul>

## Installing a Language Pack by Using the Language Pack Installer

Use the language pack installer to install a language pack for a new language. You run the command to run the installer by opening a command prompt in the *PolicyCenter/bin* directory.

### To install a language pack

1. Stop the PolicyCenter application server and Guidewire Studio, if either is running.
2. Copy the ZIP file that contains the language pack to the root of the PolicyCenter home directory.
3. At a command line prompt, navigate to:


```
PolicyCenter/bin
```

4. Enter the following command:

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

The command requires you to specify the parameters described previously in “About the Language Pack Installer” on page 24.

The language pack installer records all file operations that it executes in the command line window and in log files.

5. Restart the application server, and restart Studio if needed.
6. In PolicyCenter, the language that you installed is now a choice from the Options  menu.

## Installed Language Pack Files in the File System

After you install a language pack, the PolicyCenter *modules* directory contains the following:

- A copy of the language pack ZIP file. For example:  
pc-lang-fr.zip
- A language module folder with the same name as the language pack ZIP file. For example:  
pc-lang-fr
- An archive file that contains copies of all the files that the language pack installation modified during the installation process. For example:  
pc-lang-fr\_config\_archive-8.0.2.62.zip
- A *base.zip* file



If you restart Guidewire Studio, you can see the new module in the **Project** window. The module is in a folder with the same name as the ZIP file for the language pack. For example, if you install a language pack named `pc-lang-zh-cn`, the installer creates the following folder for it:

```
modules/pc-lang-zh-cn
```

The new folder is visible at the top level of the **Project** window in Studio, at the same level as **configuration**. The files installed by a language pack can vary.

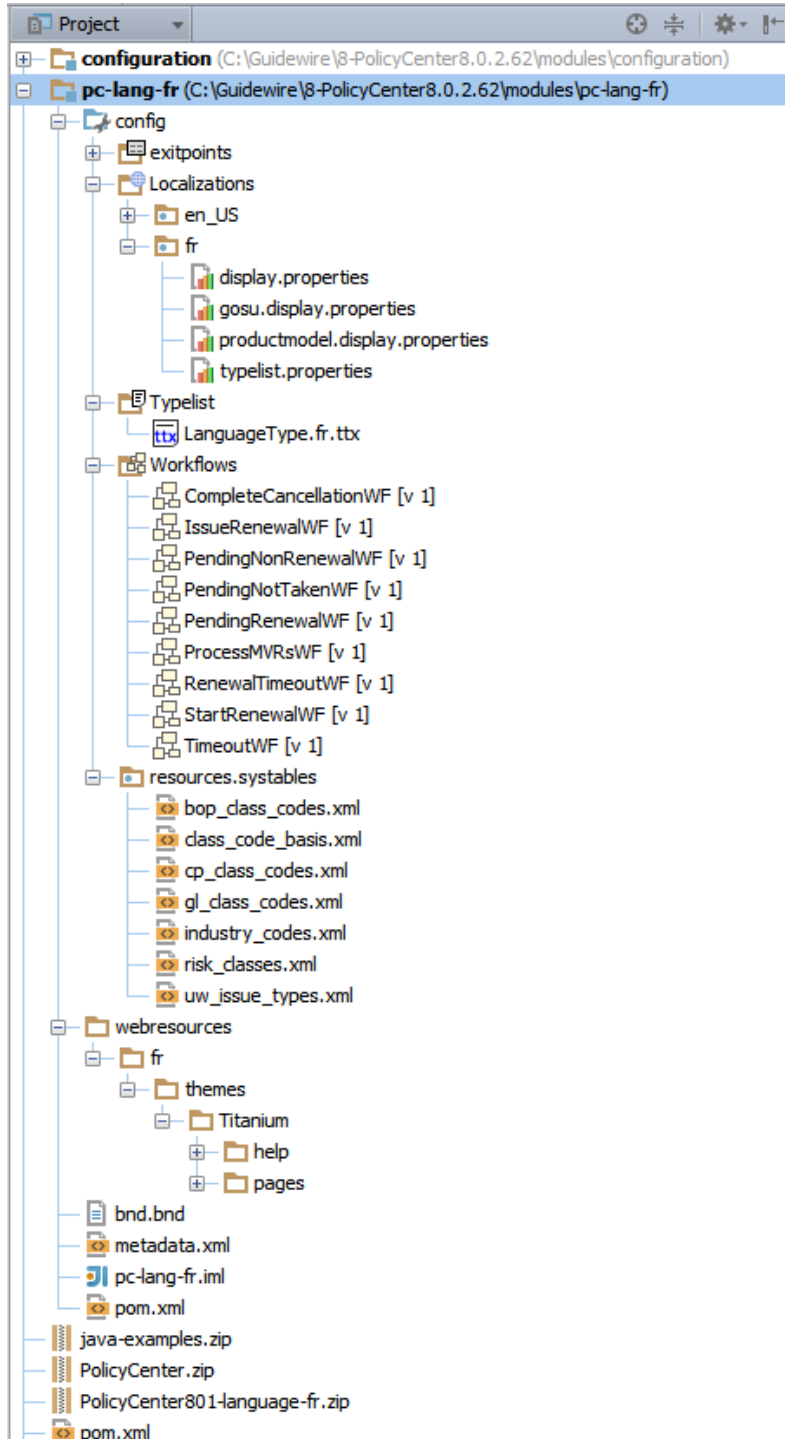
See “Installed Language Pack Files in Studio” on page 25.

## Installed Language Pack Files in Studio

After installing a language pack, if you open Guidewire Studio, you see the new language files in the **Project** window. They are stored in a folder that uses the name of the language pack. The folder is at the same level as the **configuration** folder.

**Note:** Do not edit the files in the language folder. If you want to make changes to the translations provided in these files, do so in equivalent localization folders in **configuration** → **config** → **Localizations**.

The following figure shows the files visible in the Project window of Studio after you install a French language pack in PolicyCenter:



**Note:** After you complete the language pack installation, it is possible to delete the source language pack ZIP file that you saved in the root application directory. However, do not delete any files in the `modules` directory that the language pack put there during the installation process. For example, do not delete either of the language ZIP files (the source and archive file) or the `base.zip` file from the `modules` directory.

**See also**

- “Setting the Default Display Language” on page 29
- “Selecting Language and Regional Formats in PolicyCenter” on page 13

## Activity Logging by the Language Pack Installer

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

```
11:18:07,913 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:
pc-lang-es_config_archive-8.0.2.dev.201306181134.zip
11:18:11,140 INFO ARCHIVE, source: configuration/pom.xml, target:
pc-lang-es_config_archive-8.0.2.dev.201306181134.zip
11:18:11,140 INFO ARCHIVE, source: C:/tmp/localized-module-test/pom.xml, target:
pc-lang-es_config_archive-8.0.2.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/pc-lang-es.zip/pc-lang-es, target: pc-lang-es
11:18:11,141 INFO
...
```

The first line of the log output 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: pc-lang-es/config/locale/es_PE/gosu.display.properties
...
```

In the previous example log:

- Line 1 indicates the type of step, which, in this case, is UPGRADE.
- Line 2 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 that there was an issue during the upgrade process due to a denial of file access. The log indicates 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: pc-lang-es, target: pc-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: pc-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/pc-lang-es-upgrade.zip/pc-lang-es/config/locale
/es_ES/newGuidewireSetting.properties, target: pc-lang-es/config/locale/es_ES
/newGuidewireSetting.properties
15:52:18,444 INFO COPY_TO_STAGING, source: C:/tmp/pc-lang-es-upgrade.zip/pc-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/pc-lang-es-upgrade.zip/pc-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/pc-lang-es-upgrade.zip/pc-lang-es/config/locale
/es_PE/newConfigFile.xml, target: pc-lang-es/config/locale/es_PE/newConfigFile.xml
15:52:18,445 INFO COPY_TO_STAGING, source: C:/tmp/pc-lang-es-upgrade.zip/pc-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/pc-lang-es-upgrade.zip/pc-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/pc-lang-es-upgrade.zip/pc-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/pc-lang-es-upgrade.zip/pc-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\pc-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: pc-lang-es-module-archive.zip, target: pc-lang-es
15:52:18,449 INFO

```

## Upgrading Display Languages

For information on upgrading display languages, see the release notes for your language pack.

## Setting the Default Display Language

You must install the language that you want to be your application default and set `DefaultApplicationLanguage` in `config.xml` before you start your PolicyCenter server for the first time.

---

**IMPORTANT** 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. The language pack installer adds a typecode for the installed language automatically to the `LanguageType` typelist.

**See also**

- “Globalization Parameters” on page 56 in the *Configuration Guide*

## Selecting a Personal Language Preference

Users of PolicyCenter 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” on page 29
- “Selecting Language and Regional Formats in PolicyCenter” on page 13



# Localized Printing

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

This topic includes:

- “Printing Specialized Character Sets and Fonts” on page 31
- “Localized Printing in a Windows Environment” on page 32
- “Localized Printing in a Linux Environment” on page 34

## 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, such as Japanese.

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 PolicyCenter for the font that you installed.

**See also**

- For additional information on the Apache Formatting Objects Processor, refer the following:  
<http://xmlgraphics.apache.org/fop/trunk/fonts.html>
- For Japanese fonts, refer to the following:  
<http://connie.slackware.com/~alien/slackbuilds/sazanami-fonts-ttf/pkg/12.0/>
- For information on configuring Apache FOP and PolicyCenter for specific fonts, see:
  - “Localized Printing in a Windows Environment” on page 32
  - “Localized Printing in a Linux Environment” on page 34

## Localized Printing in a Windows Environment

Suppose 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 describes how to configure Apache (FOP) and Guidewire PolicyCenter to print documents in Russian by using Cyrillic characters in a Windows environment.

- “Step 1: Download and Install Apache FOP on Windows” on page 32
- “Step 2: Configure TTFReader” on page 33
- “Step 3: Generate FOP Font Metrics Files” on page 33
- “Step 4: Register the Fonts with Apache FOP” on page 33
- “Step 5: Register Your Apache FOP Configuration and Font Family with PolicyCenter” on page 34
- “Step 6: Test Your Configuration” on page 34

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 PolicyCenter application.
- You have installed the proper language pack 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>



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

Essentially, you are changing `org.apache.fop.cli.Main` to `org.apache.fop.fonts.apps.TTFReader`. You need to do this step so that the code to generate the font metrics works correctly.

## Step 3: Generate FOP Font Metrics Files

You must generate font metrics for FOP to enable it use a font.

**To generate the font metrics:**

1. Create the following directory:  
C:\fopconfig
2. Run the following commands, which 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
```

Running these commands 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.

**To create a user configuration file for FOP, perform the following steps:**

1. Copy the following file into the directory C:\fopconfig\  
C:\fop\_install\conf\fop.xconf
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-url="arial.ttf">
  <font-triplet name="Cyrillic" style="normal" weight="normal"/>
</font>
<font metrics-file="c:\\fopconfig\\arialbd.xml" kerning="yes" embed-url="arialbd.ttf">
  <font-triplet name="Cyrillic" style="normal" weight="bold"/>
</font>
<font metrics-url="c:\\fopconfig\\ariali.xml" kerning="yes" embed-url="ariali.ttf">
  <font-triplet name="Cyrillic" style="italic" weight="normal"/>
</font>
<font metrics-file="c:\\fopconfig\\arialbi.xml" kerning="yes" embed-url="arialbi.ttf">
  <font-triplet name="Cyrillic" style="italic" weight="bold"/>
</font>
</font>
...
```

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

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

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

1. Open PolicyCenter Studio and press `Ctrl+Shift+N`, and then search for `config.xml`.
2. Open `config.xml` and set the following parameters:

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

3. Stop and start the PolicyCenter server so that these changes can take effect.

## Step 6: Test Your Configuration

After you perform the listed configuration steps, 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 PolicyCenter implementation that supports the Russian locale.

## Localized Printing in a Linux Environment

The following example illustrates how to configure Guidewire PolicyCenter 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” on page 35
- “Step 2: Download and Install Apache FOP on Linux” on page 35
- “Step 3: Configure the Font” on page 35
- “Step 4: Register the Font with Apache FOP” on page 35
- “Step 5: Register Your Apache FOP Configuration and Font Family with PolicyCenter” on page 36
- “Step 6: Test Your Configuration” on page 36

The example also assumes the following:

- You have a working Guidewire PolicyCenter 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.

To print Japanese characters, for example, you need to install a font that supports Japanese characters. The following are examples of fonts that support the printing of Japanese characters:

- IPA Gothic
- Sanazami

2. This step depends on which package manager you are using:

- If you are using the yum package manager, substitute the actual font name for *FONT-NAME* in the `yum install` command. For example:

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

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

### See also

- “Printing Specialized Character Sets and Fonts” on page 31

## 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>

2. Unpack the ZIP file into the desired directory, using your version of FOP in place of fop-0.95:

```
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 and FOP version 0.95:

```
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 version 0.95.

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 <font> section:

**Note:** Use the version of FOP that you installed in place of fop-0.95.

```
<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 PolicyCenter

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

1. Open PolicyCenter 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 PolicyCenter.

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/ <b>Note:</b> Use the version of FOP that you installed in place of fop-0.95.

3. Stop and start the PolicyCenter 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 file that uses the correct font. To test the Apache FOP configuration, you must have a PolicyCenter implementation that supports the Japanese locale.

# Localizing PolicyCenter String Resources

This topic describes how to localize the following string resources that PolicyCenter 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
- **Gosu error messages** – Strings to display as Gosu error and warning messages in Studio
- **Workflow step names** – Strings to display as individual step names in workflow processes
- **Product model string resources** – Strings to display in screens that use product models

**Note:** Ruleset names and descriptions are not localized as strings. See “Localizing Rule Set Names and Descriptions” on page 49.

This topic includes:

- “Understanding String Resources” on page 38
- “Exporting and Importing String Resources” on page 39
- “Localizing Display Keys” on page 41
- “Localizing Typecodes” on page 43
- “Localizing Gosu Error Messages” on page 45
- “Localizing Product Model String Resources” on page 46

## See also

- “Localizing Guidewire Workflow” on page 57

## Understanding String Resources

Guidewire designs PolicyCenter 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
- **Workflow Step Names** – Strings to display as individual step names in workflow processes
- **Product model string resources** – Strings to display in screens that use product models

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

This topic includes:

- “Display Keys” on page 38
- “Typecodes” on page 38
- “Workflow Step Names” on page 39

### See also

- “Localizing Product Model String Resources” on page 46

## Display Keys

PolicyCenter stores as *key/value* pairs the United States 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
```

PolicyCenter stores the key/value pairs for display keys in a file called `display.properties`. In the base configuration, PolicyCenter contains a single copy of `display.properties`, with string resources in United States English. In Guidewire Studio, you can navigate to this `display.properties` file in the **Project** window as follows:

`configuration` → `config` → `Localizations` → `en_US`

If you install a language pack for a language other than U.S. English, the installer adds a version of `display.properties` localized for that language.

### See also

- “Localizing Display Keys” on page 41.
- “Installing Display Languages” on page 23.

## Typecodes

PolicyCenter 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 in standard Java properties syntax. For example:

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

PolicyCenter stores the key/value pairs for typecodes in a file called `typelist.properties`. In the base configuration, PolicyCenter contains a single copy of `typelist.properties`, with string resources in U.S. English. In Guidewire Studio, you can navigate to the `typelist.properties` file in the **Project** window, as follows:

`configuration` → `config` → `Localizations` → `en_US`

If you install a language pack for a language other than U.S. English, the installer adds a version of `typelist.properties` localized for that language.

**See also**

- “Localizing Typecodes” on page 43.
- “Installing Display Languages” on page 23.

## Workflow Step Names

In PolicyCenter, 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 57
- “Localizing Workflow Step Names” on page 58

## Exporting and Importing String Resources

PolicyCenter enables you to export some 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
- **Workflow Step Names** – Strings to display as individual step names in workflow processes

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

Command	Related topic
<code>gwpc export-110ns [-Dexport.file] [-Dexport.locale]</code>	“Exporting Localized String Resources with the Command Line Tool” on page 39
<code>gwpc import-110ns [-Dimport.file] [-Dimport.locale]</code>	“Importing Localized String Resources with the Command Line Tool” on page 40

The commands provide parameters that enable you to specify the locations of the export and import files 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**

- For information on using Java property files in PolicyCenter, see “Properties Files” on page 393 in the *Gosu Reference Guide*.

## Exporting Localized String Resources with the Command Line Tool

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

- Display keys
- Typecodes
- Workflow step names

The export file organizes the strings into translated and non-translated groups. The command provides parameters that enable you to 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:

```
PolicyCenter/bin
```

3. Run the following command:

```
gwpc export-110ns -Dexport.file=targetFile -Dexport.locale=localizationFolder
```

#### -Dexport.file Command Parameter

Command line parameter `-Dexport.file` specifies *targetFile*, the name of the file in which PolicyCenter saves the exported resource strings. You must add the file extension to the file name. By default, PolicyCenter puts the export file in the root of the installation directory. You specify the directory as follows:

- To leave the export file in the same location, enter only the name of the file to export.
- To save 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 *localizationFolder*, the target localization folder for the translated strings. The localization folder name must match a PolicyCenter language type that exists in the `LanguageType` typelist. For example: `fr_FR` or `ja_JP`.

#### See also

“Importing Localized String Resources with the Command Line Tool” on page 40

## Importing Localized String Resources with the Command Line Tool

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

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

The command provides parameters that enable you to 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:

```
PolicyCenter/bin
```

3. Run the following command:

```
gwpc import-110ns -Dimport.file=sourceFile -Dimport.locale=localizationFolder
```



**-Dimport.file Command Parameter**

Command line parameter `-Dimport.file` specifies *sourceFile*, the file that contains the translated resource strings. It must be in the same format as a file exported from PolicyCenter. By default, PolicyCenter puts the export file in the root of the installation directory. You can set the directory as follows:

- To leave the import translation file in the same location, you need enter only the name of the file to import.
- To move the translation file to a different location, you must enter an absolute or relative path to the file from the root of the installation directory.

**-Dimport.locale Command Parameter**

Command line parameter `-Dexport.locale` specifies *localizationFolder*, the name of the localization folder into which to save the translated strings. The localization folder name must match a PolicyCenter language type that exists in the `LanguageType` typelist, such as `fr_FR` or `ja_JP`.

**See also**

“Exporting Localized String Resources with the Command Line Tool” on page 39

## Localizing Display Keys

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

In the base configuration, PolicyCenter provides a single `display.properties` file. In the Studio **Project** window, this file is located in:

`configuration` → `config` → `Localizations` → `en_US`

The **Localizations** node contains multiple folders with localization names such as `de_DE`, each of which can also contain, after configuration, additional property files. In addition, any language pack that you install also contains property files.

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

Translation technique	Related topic
Using the Studio <b>Display Keys</b> editor	“Localizing Display Keys by Using the Display Key Editor” on page 42
Using the display key import and export tools	“Exporting and Importing String Resources” on page 39

**See also**

- For more information on display keys and string resources in general, see “Display Keys” on page 38.

## PolicyCenter and the Master List of Display Keys

Using the localization node property files, on startup, PolicyCenter generates a master list of display keys for use in the user interface. For each property file, PolicyCenter 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 PolicyCenter logs a warning message noting that it found a display key value with different arguments in different regions. For example:

Configuration Display key found with different argument lists across locales: `Validator.Phone`

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

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

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

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

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

## Localizing Display Keys by Using the Display Key Editor

It is possible to enter a localized version of a display key directly in the Studio editor. To access the editor, first, in Studio, navigate to **configuration** → **config** → **Localizations**. Expand the node for the target language and open the `display.properties` file in that folder. You can also press `Ctrl+Shift+N` to find a property file for a specific region.

It is not necessary to use Studio to localize display keys. If you have a large number of translated strings to enter, you can use the export and import commands. With these commands, you export the strings, translate them, and import the translated strings into Studio. See “Exporting and Importing String Resources” on page 39.

## Identifying Missing Display Keys

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

- Compares each language configured on the server against the master display key list.
- 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:

```
gwpc displaykey-diff
```

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

```
PolicyCenter/build
```

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

- The tool creates a subdirectory for that language using the localization 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 localization 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 `en_US`:

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

**Note:** PolicyCenter does not generate a `display.properties` file for a region that does not have any missing display keys.

## Working with Display Keys for Later Translation

It is possible to create a display key in a specific localization folder 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. Use the same string in all cases to make it easy to find the placeholder display keys. Using a string such as [TRANSLATE] makes it easy to see the string in the PolicyCenter 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 the placeholder strings so that you do not miss any during a search.

## Localizing Typecodes

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

- **Using the gwpc export and import commands** – Use these commands to localize all typecodes by editing a single text file. See “Using the gwpc Export and Import Commands” on page 43.
- **Using the Typelist Localization editor** – Enter localized values for individual typecodes directly through the Typelist editor. See “Using the Typelist Localization Editor” on page 43.
- **Editing the typelist.properties file in a localization folder** – Navigate to a localization folder and open the `typelist.properties` file so you can edit it.

You can also specify a sort order for each typelist, by language. See “Setting Localized Sort Orders for Localized Typecodes” on page 44.

There is also Gosu syntax for accessing typecodes that you need to be aware of. See “Accessing Localized Typekeys from Gosu” on page 45.

## Using the gwpc Export and Import Commands

You use the `gw import` and `export` commands to create a single file and localize typecodes for all typelists in that file. The exported file has all the typecodes in it, which cannot be guaranteed for any given `typelist.properties` file. The typecodes are separated into localized and unlocalized typecodes, which makes it easier to see which ones you need to translate.

After you import the translated file, the `typelist.properties` file for the given localization code is updated.

### See also

- “Exporting Localized String Resources with the Command Line Tool” on page 39
- “Importing Localized String Resources with the Command Line Tool” on page 40

## Using the Typelist Localization Editor

**Note:** You can localize typecodes in the Typelist Localization editor only for languages that you configured in PolicyCenter. See “Installing Display Languages” on page 23.

You can use the Typelist Localization editor to localize typecodes. This technique is labor intensive, but can be useful if you want to localize just a few typecodes, or you want to localize some extension typecodes that you have created. Editing typecodes in this editor causes PolicyCenter to add them to the associated `typelist.properties` file.

**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 corner of the Typelist editor.
4. Find the group with the language code 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.

For example, you installed the French language pack, so you see the **fr** language code. Defining the French language version of the **name** field for a typecode updates the **typelist.properties** file in **configuration → config → Localizations → fr**. If necessary, PolicyCenter creates a file in this location and then adds the entry.

## Editing the **typelist.properties** file

The **typelist.properties** file in a localization folder provides localized definitions for typecodes for the language represented by the folder. This file does not necessarily have definitions for all typecodes. You might edit this file, for example, to override or add to the translations in the **typelist.properties** file saved by the language pack installer.

As you use either of the other techniques to update typecode localizations, this file is also updated.

- If you want to ensure that all typecodes are localized for a language, use export and import. See “Using the gwpc Export and Import Commands” on page 43.
- If you want to edit just a few typecodes, you can use the Typelist Localization editor. See “Using the Typelist Localization Editor” on page 43

**To edit a **typelist.properties** file**

1. Open the Studio **Project** window.
2. Navigate to the localization folder and double-click the file to open it.

For example, navigate to **configuration → config → Localizations → en\_US** and double-click **typelist.properties**.

**Note:** If there is no **typelist.properties** file in the folder, you can copy one in or right-click the folder and create a new file.

3. Edit the file, and then save it when you are finished.

## Setting Localized Sort Orders for Localized Typecodes

It is possible to set the sort order for the typecodes in a typelist by creating a file named after the typelist, with a file extension of **.sort**. You save the file in the localization folder for the language to which the sort order applies. PolicyCenter stores the sort order information, by language, in the typelist table.

**Note:** A typical use for a **.sort** file is to support Japanese with other languages on the same server. For example, you might want to provide a sort order for Japanese provinces, which customarily are in order from north to south—Hokkaido, Aomori, Iwate, and so on. Otherwise, you can prioritize typecodes by defining their priority in the typelist and in the **language.xml** files for each language. See “Determining the Order of Typekeys” on page 124.

Typecodes in the typelist that are not listed in the **.sort** file are ordered according to the sort order specified in the **language.xml** file for that region.

PolicyCenter does not provide any sort order files in the base configuration. You must put any `.sort` file that you create in the appropriate localization folder. For example, for Japanese, put 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
<code>typelist.typekey.Code</code>	String that represents the typecode
<code>typelist.typekey.DisplayName</code>	Localized language version of the entity name
<code>typelist.typekey.UnlocalizedName</code>	Name listed in the data model

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

```
var displayString = myTypekey.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 Gosu Error Messages

Similar to the key/value pairs stored in `display.properties`, PolicyCenter stores the string resources defining Gosu errors in file `gosu.display.properties`. These strings display in Guidewire Studio if there is an error in Gosu code or if code is being compiled and an error occurs. Do not add your own strings to this file.

In the base configuration, Guidewire provides only a single U.S. English version of this file. In Studio, you can see this file at the following location in the **Project** window:

configuration → config → Localizations → en\_US

File `gosu.display.properties` provides the key/value pairs like the following ones:

```
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 pack, that language pack contains a translated version of file `gosu.display.properties` in the target language.

### See also

- “Localizing Display Keys” on page 41

- “Localizing Typecodes” on page 43

## Localizing Product Model String Resources

**Note:** You manage most product model functionality through Guidewire Product Designer.

Guidewire PolicyCenter uses many of the data fields in the PolicyCenter data model to display information in the PolicyCenter interface. These data fields are string resources that symbolically represent business data configuration used in various places in Guidewire PolicyCenter. These data fields are the same key/value pairs used in other application properties files.

Guidewire stores these string resources in file `productmodel.display.properties`. By translating the contents of this file, it is possible to provide translated versions of the following:

- All product model **Name** and **Description** fields
- All question **Name** and **Description** fields
- All question **Text** and **Failure message** fields

In the base configuration, Guidewire provides only a single U.S. English version of this file, in the following location in Studio:

`configuration` → `config` → `Localizations` → `en_US`


Every additional Guidewire language pack that you install also installs another version of `productmodel.display.properties` with the string resources in the target language.

See “Localizing PolicyCenter String Resources” on page 37 for details of working with display string properties files.

This topic includes:

- “Translating Product Model Strings in Product Designer” on page 46
- “Localizing Coverage Term Options” on page 46

### Translating Product Model Strings in Product Designer

To facilitate translating product model strings in Product Designer, use the **Display Key Values by Locale** dialog box. You access this dialog box through the **Translate** icon  that appears at the end of selected product model fields. In this dialog, you can add a translated version of the string label for any defined language in PolicyCenter.

Product Designer adds any change that you make to the product model field labels to the active change list. Product Designer pushes the active change list back to the application configuration files after you commit your change. See “Working with Change Lists” on page 17 in the *Product Designer Guide* for a discussion of how change lists work in Product Designer.

For you to see the field label change in PolicyCenter, you must manually push the change list back to PolicyCenter.

### Localizing Coverage Term Options

Coverages in the PolicyCenter product model can have *coverage terms*. A coverage term is a statement or a value that defines the extent or limit of the coverage. A *coverage term option* is one of a set of coverage terms pertaining to a specific coverage. Coverage term options can be textual or numeric, but are always stored as strings. You can localize coverage term options in Product Designer.

Textual coverage terms options are descriptions that help classify a limitation or extent of the coverage. Examples of textual coverage term options include:

- Class 1 Employees

- Class 2 Employees
- Alaska Attorney Fees Limit

Numeric coverage term options include single values and packages. A *package* is a set of values selected as a single item. Examples of numeric coverage term options include:

- 10,000
- 250K
- \$20,000/\$50,000/\$10,000

The string value that users see after selecting a numeric coverage term option is stored in the **Description** of the option. The actual numeric value represented by this description is stored in the **Value** of the option. For example:

- Description = "\$12,500"
- Value = 12500

Because they are stored as strings, numeric coverage term option display values often include separators, decimal points, and currency symbols. And because they are stored as strings, these separators, decimal points, and currency symbols are not influenced by regional settings. Furthermore, the sample coverage term option values that Guidewire provides in the PolicyCenter base configuration are potentially suitable only for a North American deployment.

Therefore, to localize numeric coverage term options, you must:

- Choose coverage term option values that are suitable for your target region.
- Localize any currency symbols, separators, or decimal points to suit your target region.





# Localizing PolicyCenter with Studio

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

This topic includes:

- “Viewing Unicode Characters in Studio” on page 49
- “Localizing Rule Set Names and Descriptions” on page 49
- “Setting the IME Mode for Field Inputs” on page 50
- “Setting a Language for a Block of Gosu Code” on page 51

## Viewing Unicode Characters in Studio

To configure Guidewire Studio to display Unicode characters:

1. In Studio, navigate to **File** → **Settings** → **Appearance**.
2. Change the **Theme** setting to a value that works for your operating system.
  - If you are on Microsoft Windows, change the theme to **Windows**.
  - If you are on Linux, change the theme to **Nimbus**.
3. Click **OK**.

**Note:** Guidewire does not supply fonts for your system. You must download the fonts for the languages you want to use in Guidewire Studio. For example, for Japanese fonts, see:

<http://connie.slackware.com/~alien/slackbuilds/sazanami-fonts-ttf/pkg/12.0/>

## 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.

In PolicyCenter Studio, to access the rule sets, navigate in **Project** window to:

**configuration** → **config** → **Rule Sets**

In 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("Translated rule set name")
		Rule set description	@gw.rules.RuleSetDescription("Translated description")
.gr	Rule	Rule name	@gw.rules.RuleName("Translated rule name")

To modify these files, open them in a text editor in your system directory structure and not in Studio.

You can view only a single language translation of a rule set name or description in Studio. You cannot provide multiple translations at once, as you can with string translations.

## Setting the IME Mode for Field Inputs

*IME mode* controls the state of an input method editor (IME) for entering Japanese or Chinese language characters. PolicyCenter provides three states for `imeMode`, which you set at the field level in Guidewire Studio. The three possible states are:

State	Description
Active	<ul style="list-style-type: none"> <li>Japanese – Turns on the last entry type that you selected. For example, Hiragana or full-width Katakana,.</li> <li>Chinese – Turns on pinyin entry</li> </ul>
Inactive	<ul style="list-style-type: none"> <li>Japanese – Turns off Roman entry</li> <li>Chinese – Turns on Roman entry</li> </ul>
Not selected	Does nothing – Leaves the currently-selected IME mode as set

You set the `imeMode` at the field level on a PCF file. For a *rōmaji* field in Japanese, for example, you might set `imeMode` to `inactive`. Then, on the next field, one that needs Kanji entry, you would set `imeMode` to `active`, and so on for the various fields in the PCF file. In general, you set `imeMode` on text input fields and possibly drop-down fields such as typelists and range input fields.

It is important to understand that PolicyCenter does not actually set the actual IME mode. PolicyCenter merely turns IME on or off for each field. PolicyCenter cannot dictate that a certain field must contain Zenkaku Katakana and a different field must contain Hiragana. The choice of input conversion style is left to the user.

## IME and Text Entry

Guidewire applications support Unicode characters by default in the base configuration. However, certain PCF entry fields do not support using IME to enter Unicode characters. Many of the PolicyCenter application screens capture keystrokes as a user enters data. This capture mechanism does not work properly if you use IME to input data directly into input fields.

In particular, if you use IME to input data instead of keystrokes, the following entry field types do not process character data properly:

- Text entry fields that implement `inputMask` functionality.

- Text entry fields that implement `maxChars` functionality, especially `TextArea` input fields.

**Note:** Unlike `TextBox` fields, there is no built-in browser support for `TextArea` input fields.

## Setting a Language for a Block of Gosu Code

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

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

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

**Note:** The second method sets both region and language at the same time. This topic covers only setting the language.

A typical use of this feature is to override the current language that the Gosu code block uses by default. The default current language is specified by the current user or the `DefaultApplicationLanguage` parameter in `config.xml`. If neither is set, PolicyCenter uses the browser language setting.

The parameters for the methods are:

Parameter	Description
<code>alternateLocale</code>	An object of type <code>ILocale</code> that represents a regional format from the <code>LocaleType</code> typelist. Specify a <code>GWLocale</code> object for this parameter.
<code>alternateLanguage</code>	An object of type <code>ILocale</code> that represents a language from the <code>LanguageType</code> typelist. Specify a <code>GWLanguage</code> object for this parameter.
<code>\ -&gt; { <i>code</i> }</code>	A Gosu block as a <code>GWRunnable</code> object—the Gosu code to run in a different locale or language

### Specifying an `ILocale` Object for a Language Type

To run a block of Gosu code with a specified language, you must use a language object of type `GWLanguage` for the first parameter to `runAsCurrentLanguage`. You can specify the object in a number of ways:

- Use the `gw.api.util.LocaleUtil.toLanguage` method to provide a `GWLanguage` object that corresponds to a Gosu typecode in the `LanguageType` typelist. You can specify the parameter to this method by using typecode syntax, such as `LanguageType.TC_EN_US`. The typecode has to be defined in the `LanguageType` typelist.
- You can specify a typecode of the correct type directly, without using the `toLanguage` method. For example, for U.S. English, use `gw.i18n.ILocale.LANGUAGE_EN_US`. This syntax requires that the language typecode `en_US` be defined in the `LanguageType` typelist.
- You can specify the first parameter by using a method on `LocaleUtil` that can get the current or default language. For example, `getCurrentLanguage` and `getDefaultLanguage`.

You can add a typecode to the `LanguageType` typelist. If you add a new typecode to this typelist, you must restart Studio to be able to use it in `gw.i18n.ILocale`. For a similar example, see “To add a Java locale code to the `LocaleType` typelist” on page 72.

The following example Gosu code sets U.S. English as the language for a display string, overriding the current language:

```
uses gw.api.util.LocaleUtil

// Run a block of Gosu code that prints the display string in U.S. English
LocaleUtil.runAsCurrentLanguage(
    LocaleUtil.toLanguage(LanguageType.TC_EN_US),
    \-> {print(displaykey.Activity)})
```

**See also**

- “Setting Regional Formats for a Block of Gosu Code” on page 75
- “Gosu Blocks” on page 231 in the *Gosu Reference Guide*

**Additional Useful Methods on `gw.api.util.LocaleUtil`**

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

Method	Returns...
<code>canSwitchLanguage</code>	Boolean <code>true</code> if the current user is allowed to switch to a different language
<code>canSwitchLocale</code>	Boolean <code>true</code> if the current user is allowed to switch to a different locale
<code>getAllLanguages</code>	List of all available languages as defined in the <code>LanguageType</code> typelist
<code>getAllLocales</code>	List of all available locales as defined in the <code>LocaleType</code> typelist
<code>getCurrentLanguage</code>	Current language, which can be based on the user language setting
<code>getCurrentLocale</code>	Current locale, which can be based on the user locale setting
<code>getCurrentLanguageType</code>	Language set for the current user
<code>getCurrentLocaleType</code>	Locale set for the current user
<code>getDefaultLanguage</code>	Language set by configuration parameter <code>DefaultApplicationLanguage</code>
<code>getDefaultLocale</code>	Locale set by configuration parameter <code>DefaultApplicationLocale</code>
<code>getLanguageLabel</code>	Language as a <code>String</code> value, given a language type
<code>getLocaleLabel</code>	Locale as a <code>String</code> value, given a locale type
<code>toLanguage</code>	Sets language, given a language typecode
<code>toLocale</code>	Sets locale, given a locale typecode

# Localizing Administration Data

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

This topic includes:

- “Understanding Administration Data” on page 53
- “Localized Columns in Entities” on page 53
- “Localization Tables in the Database” on page 55
- “System Table Localization” on page 55

## Understanding Administration Data

Guidewire refers to certain types of application data to as *administration data*, or by the shortened term *admin data*. For example, activity patterns are administration data. For selected fields in administration data, PolicyCenter stores localized—translated—values directly in the application database.

You enter translations for admin data directly in PolicyCenter, in screens that you can open on the **Administration** tab. If you have configured PolicyCenter for multiple languages, for entities with localization tables, you see a **Localization** list view that is visible at the bottom of a screen. This list view has a row for each enabled language in the application and has fields for each element on that screen that you can localize.

## Localized Columns in Entities

To accommodate localized values for shared administration data or any entity data, you can specify that a column contains localized values in the database. To configure an entity to store localized values for a column, in Guidewire Studio, add a `Localization` element as a child of the `column` element for the entity.

For example, in Studio open the entity in the editor. Then right-click the column you want to localize and choose **Add new → localization**.

Adding a `localization` element to a column triggers the creation of a localization table during the next database upgrade. A *localization 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 requires that you specify a `tableName` attribute, which is the name of the localization table. This name has length restrictions and a special format. For an example, see “Localization Tables in the Database” on page 55.

## Localization Attributes

Guidewire provides several attributes on the `localization` element on column that affect the use of the element. The following list describes each attribute:

Attribute	Type	Description
<code>nullok</code>	Boolean	<p>This attribute is required. Set it to the same value as the <code>nullok</code> attribute for the column to which you are adding the table.</p> <p>If you set this attribute to <code>false</code>, PolicyCenter flags missing entries that it finds during a database consistency check, but it can start up with these missing entries. If PolicyCenter is configured with multiple languages:</p> <ul style="list-style-type: none"> <li>PolicyCenter stores the values for the default application language in the main database table of the entity.</li> <li>PolicyCenter stores the values for additional languages in a separate localization table.</li> </ul> <p>During a consistency check, PolicyCenter flags entries in the main database table for the default language if corresponding entries for additional languages cannot be found in the localization table. Entries flagged as missing additional languages are warnings only. A missing language value does not prevent the server from starting.</p> <p><b>Note:</b> If only one language is configured, PolicyCenter does not run the consistency check.</p>
<code>tableName</code>	String	<p>The name of the localization table. Use the following format for this name:</p> <p><i>mainEntityNameAbbreviation_columnNameAbbreviation_110n</i></p> <p><b>IMPORTANT:</b> The table name must be no longer than 16 characters. If the name exceeds this length, the application server will not start.</p>
<code>extractable</code>	Boolean	<p>Default value is <code>false</code>. If you set this attribute to <code>true</code>, PolicyCenter adds the localization table to the archive for the entity. See “The Extractable Delegate” on page 251 in the <i>Configuration Guide</i>.</p>
<code>overlapTable</code>	Boolean	<p>Default value is <code>false</code>. Overlap tables are tables in which individual table rows can exist either in the domain graph or as part of reference data, but not both. The database table itself exists both in the domain graph and as reference data. If you set this attribute to <code>true</code>, PolicyCenter marks the localization table as an overlap table. See “The OverlapTable Delegate” on page 251 in the <i>Configuration Guide</i>.</p>
<code>unique</code>	Boolean	<p>Default value is <code>false</code>. If you set this attribute to <code>true</code>, PolicyCenter enforces that for each language the values are unique and there are no duplicates.</p> <p>If the entity is of type <code>effdated</code> or <code>effdatedbranch</code>, do not set <code>unique</code> to <code>true</code>. See “Data Entities and the Application Database” on page 155 in the <i>Configuration Guide</i>.</p>

### See also

- “Checking Database Consistency” on page 38 in the *System Administration Guide*

## Localization Element Example

In the base configuration, the `ActivityPattern` entity’s `Description` column is configured for localization. The `Description` column can store localized values for each configured language.

**To see the localization element definition on the Description column**

1. In Studio, navigate in the Project window to configuration → config → Metadata and double-click Activity.eix to open it in the editor.
2. Expand the Description column, and then click its localization element.
3. This element has the following property values:

Property	Value
nullok	true
tablename	actpat_desc_110n
extractable	false
overlapTable	false
unique	false

## Localization Tables in the Database

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

*mainEntityNameAbbreviation\_columnNameAbbreviation\_110n*

---

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

---

For example, the localization table name for the Subject column of the ActivityPattern entity uses the abbreviation actpat for the main entity and sbj for the column name:

**actpat\_sbj\_110n**

Localization tables have 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

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

## System Table Localization

System tables are database tables that support business logic in PolicyCenter lines of business. Developers who use Guidewire Studio define system tables with needed columns as entities in the data model. Business analysts who use Product Designer can then examine, edit, and enter values for the system tables columns.

System tables provide additional metadata beyond the capacity of typelists. System tables typically provide storage for information that must be maintained periodically by non-developers. Examples include:

- Class codes
- Territory codes
- Industry codes
- Reason codes
- Reference dates
- Rate factors


- Underwriting companies

#### See also

- For complete information, see “System Tables” on page 75 in the *Product Model Guide*.

## The Product Designer System Table Editor

In Product Designer, you can examine, edit, and enter values for system table columns. In addition, you can enter localized values for localized system tables columns.

In Product Designer, the **System Table** page displays content for localized columns in the language that you select in the **User Settings** page. If a localized column has a value for that language, the column displays it in the **System Table** page. If a localized column does not have a value for that language, the column is empty. To examine, enter, or edit values for other languages, click the Translate  button to display the **Display Key Values by Locale** dialog box.

#### See also

- To learn how to configure a column as a localized column, see “Localized Columns in Entities” on page 53.
- To learn how the database stores localized values, see “Localization Tables in the Database” on page 55.

## What You Can Localize

The default configuration of PolicyCenter provides localized columns in the following system tables.

System table file	System table entity	Localized column
bop_class_codes.xml	BOPClassCode	• Classification
class_code_basis.xml	ClassCodeBasis	• Name • Description
cp_class_codes.xml	CPCClassCode	• Classification
gl_class_codes.xml	GLClassCode	• Classification
industry_codes.xml	IndustryCode	• Classification
risk_classes.xml	RiskClass	• Description
territory_codes.xml	DBTerritory	• Description
wc_class_codes.xml	WCCClassCode	• Classification • ShortDesc



# Localizing Guidewire Workflow

This topic discusses localization as it relates to Guidewire workflow.

This topic includes:

- “Localizing PolicyCenter Workflow” on page 57
- “Localizing Workflow Step Names” on page 58
- “Creating a Locale-Specific Workflow SubFlow” on page 59
- “Localizing Gosu Code in a Workflow Step” on page 59

## Localizing PolicyCenter Workflow

At the start of the execution of a workflow, PolicyCenter evaluates the language and locale set for the workflow. PolicyCenter then uses that language for notes, documents, templates, and similar items associated with the workflow. The language and locale that the workflow uses depend on the settings of the user that executes the workflow code.

**Note:** See “Localizing Templates” on page 61 for information on localizing application documents, notes, and emails.

It is possible to set the workflow region and the workflow language independently of the default application language and region in Studio. To set either workflow language or region, open the workflow in the Studio Workflow editor. Navigate in the **Project** window to **configuration** → **config** → **Workflows** and double-click the workflow node to open it in the Studio Workflow editor. Then click 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	<code>gw.i18n.ILocale.FR_FR</code>
Variable expression	<code>gw.api.util.LocaleUtil.toLocale(PolicyPeriod.Policy.Account.PrimaryLanguage)</code> <code>gw.api.util.LocaleUtil.toLocale(Group.Supervisor.Contact.PrimaryLanguage)</code>

## Localizing Workflow Step Names

You can provide translated versions of workflow step names. PolicyCenter 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 PolicyCenter by selecting a language from the **Language** submenu on the **Options**  menu.

You can provide translated workflow steps names as described in the following topic:

- “Exporting Workflow Steps Names as String Resources for Translation” on page 58

### See also

“Selecting Language and Regional Formats in PolicyCenter” on page 13

## Exporting Workflow Steps Names as String Resources for Translation

### To export workflow step names as string resources for translation

1. Export the PolicyCenter string resources for a particular locale by using the following `gwpc` command syntax:

```
gwpc export-110ns -Dexport.file=targetFile -Dexport.locale=localizationFolder
```

In the command, you must provide a target file name and specify from which localization folder to export the string resources.

2. Find the workflow by name. For example:

In PolicyCenter, find `Workflow.ProcessMVRsWF` and keep searching until you see an entry that ends in `.Step`. For example, `Workflow.ProcessMVRsWF.1.BeforeOrder.Step`.

**Note:** The display key names for workflow items might contain a box character between the workflow name and the version number. A box character represents a character that your system is not configured to display, in this case the character *one dot leader*, UTF-16 hexadecimal code 0x2024. The previous examples represent that character as a period.

3. Translate the workflow step names into the target language.
4. Import the translated strings resources back into PolicyCenter by using the following `gwpc` command syntax:

```
gwpc import-110ns -Dimport.file=sourceFile -Dimport.locale=localizationFolder
```

In the command, you must provide a source file name and specify into which localization folder to import the string resources.

### See also

- For command parameters, “Exporting and Importing String Resources” on page 39.
- 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
<code>createSubFlow</code>	Creates a child subflow synchronously, meaning that PolicyCenter 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.
<code>createSubFlowAsynchronously</code>	Creates a child subflow asynchronously, meaning that PolicyCenter 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.  Because PolicyCenter 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.
<code>instantiateSubFlow</code>	Creates a child subflow, but does not start it. 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 393 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 PolicyCenter Workflow” on page 57 for details.
3. Instantiate the subflow by 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 by using one of the workflow start methods described at “Instantiating a Workflow” on page 396 in the *Configuration Guide*.

### See also

“Workflow Subflows” on page 402 in the *Configuration Guide*

## Localizing Gosu Code in a Workflow Step

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

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

In the code, set the value of `alternateLanguage` to the language to use for the Gosu code block. For example:

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

Other wrapper methods useful for localization include the following:

- `runAsCurrentLocale`
- `runAsCurrentLocaleAndLanguage`

For more information on these methods, see:

- “Setting a Language for a Block of Gosu Code” on page 51
- “Setting Regional Formats for a Block of Gosu Code” on page 75



# Localizing Templates

This topic describes application localization as it applies to documents, emails, and notes.

This topic includes:

- “About Templates” on page 61
- “Creating Localized Documents, Emails, and Notes” on page 62
- “Document Localization Support” on page 66

## About Templates

In the base configuration, Guidewire provides a number of template-related definition files for notes, emails, and documents. You can navigate in the Studio **Project** window to the following folders containing these files:

- **configuration** → **config** → **resources** → **doctemplates**
- **configuration** → **config** → **resources** → **emailtemplates**
- **configuration** → **config** → **resources** → **notetemplates**

Each folder contains two files for each resource type, the template file and a descriptor file. The two files have the same names but different file extensions. PolicyCenter uses the files to define a document, an email, or a note. The following table describes these files.

File extension	Description	Example
.rtf .htm .pdf .xml .xls	Template files of various types. Template files contain the actual content of the document, email, or note.	CreateEmailSent.gosu.htm
.descriptor	<p>Template descriptor file in XML format. This file contains the template metadata, such as:</p> <ul style="list-style-type: none"> <li>• name</li> <li>• subject</li> </ul> <p>This file can also contain symbol definitions for context objects that PolicyCenter substitutes into the template content file in creating the final document.</p>	CreateEmailSent.gosu.htm.descriptor

#### See also

For general information on templates and how to create them and use them, see:

- “Gosu Templates” on page 347 in the *Gosu Reference Guide*
- “Data Extraction Integration” on page 637 in the *Integration Guide*

## Creating Localized Documents, Emails, and Notes

Creating localized versions of document, email, and note templates mainly involves:

- Creating locale-specific folders in the correct location.
- Populating each folder with translated versions of the required document, email, or note templates and descriptor files.

The following steps describe the process.

- Step 1: Create Locale-Specific Folders
- Step 2: Copy Template Content Files
- Step 3: Localize Template Descriptor Files
- Step 4: Localize Template Files
- Step 5: Localize Documents, Emails, and Notes in PolicyCenter

#### Notes:

- In PolicyCenter, the default locale for a document, note, or email template is the configured default locale for the application.
- Any time you add a file to a Studio-managed file folder, you must stop and restart Studio so that it recognizes the change.
- Guidewire does not provide the ability to localize Velocity templates.

### Step 1: Create Locale-Specific Folders

In PolicyCenter Studio, you can see the locations of the unlocalized PolicyCenter document, email, and note templates by navigating in the **Project** window to the following folders:

- **configuration** → **config** → **resources** → **doctemplates**
- **configuration** → **config** → **resources** → **emailtemplates**

- `configuration → config → resources → notetemplates`

#### To create your own localized versions of the template files:

1. Open Studio and navigate in the **Project** window to the following folder:  
`configuration → config → resources → doctemplates`
2. Right-click the **doctemplates** folder and choose **New → Package**.
3. Enter the name of your locale-specific folder and click **OK**.  
For example, if you are creating a French locale and want to use French-language document templates, enter `fr_FR` for the name.
4. If you also want to localize the email and note templates, repeat the previous steps for the following folders:
  - `configuration → config → resources → emailtemplates`
  - `configuration → config → resources → notetemplates`

After you complete this task, you see the locale-specific folders in the **Project** window, along with all the non-localized templates. For example:

- `configuration → config → resources → doctemplates → fr_FR`
- `configuration → config → resources → emailtemplates → fr_FR`
- `configuration → config → resources → notetemplates → fr_FR`

## Step 2: Copy Template Content Files

After you set up the template locale folders, copy the template files from the main directory into the locale subfolders.

- For documents, you copy only the template content files, not the descriptor files. For example, you might copy the following files from **doctemplates** to **doctemplates/fr\_FR**:  

```
CreateEmailSent.gosu.htm
PolicyQuote.gosu.rtf
```
- For email and notes, you copy both the template content files and the template descriptor files. For example, you might copy the following files from **emailtemplates** to **emailtemplates/fr\_FR**:

```
NeedXYZ.gosu
NeedXYZ.gosu.descriptor
GotXYZForPolicy.gosu
GotXYZForPolicy.gosu.descriptor
ActivityActionPlan.gosu
ActivityActionPlan.gosu.descriptor
```

## Step 3: Localize Template Descriptor Files

After you copy the template files to a template locale folder, you create localized versions of the template descriptor files.

It is important to understand that localizing template descriptor files serves a different purpose from that of localizing (translating) template content files. For example:

- Localizing the subject context object in an email template descriptor file enables a PolicyCenter user to see the subject line of that email template in the localized language in PolicyCenter.
- Localizing the content of an email template enables the recipient of that email to see its contents in the localized language.

The manner in which you create localized document template descriptor files is different from the process for creating localized email and note template descriptor files. See the following topics for details:

- [Localizing Document Descriptor Files](#)
- [Localizing Email and Note Descriptor Files](#)

## Localizing Document Descriptor Files

The document template descriptor files remain in the main **doctemplates** folder, and you edit them there. Descriptor files are XML-based files that conform to the specification defined in file `document-template.xsd`. You can view these files in the Studio **Project** window at:

**configuration** → **config** → **resources** → **doctemplates**

The descriptor file defines context objects, among other items. *Context objects* are values that PolicyCenter inserts into the document template to replace defined symbols. For example, PolicyCenter replaces `<%=Subject%>` in the document template with the value defined for the symbol `Subject` in the descriptor file.

For example, in the base configuration, PolicyCenter provides an XML definition for the `AccountEmailSent` template descriptor associated with the `AccountEmailSent` document content template. The descriptor file defines a context object for the `Subject` symbol. You can define as many context objects and associated symbols as you need. You can add elements that localize the template for any languages supported by your system.

```
<?xml version="1.0" encoding="UTF-8"?>
<DocumentTemplateDescriptor
  id="AccountEmailSent.gosu.htm"
  name="Gosu Sample Account Email Sent Record"
  description="Record of an email being sent"
  ...
  keywords="CA, email">

  <DescriptorLocalization language="someLanguage" name="localizedName"
    description="localizedDescription" />
  ...

  <ContextObject name="Subject" type="string">
    <DefaultObjectValue></DefaultObjectValue>
    <ContextObjectLocalization language="someLanguage" display-name="localizedName" />
    ...
  </ContextObject>
  ...
</DocumentTemplateDescriptor>
```

Localizing a template descriptor file requires that you localize a number of items in the file. The following list describes some of the main items that to localize in a descriptor file:

Element	Attribute	Description
<code>&lt;DocumentTemplateDescriptor&gt;</code>	<ul style="list-style-type: none"> <li><code>keywords</code></li> </ul>	Localize the keywords associated with this template to facilitate the search for this template in the PolicyCenter search screen.
<code>&lt;DescriptorLocalization&gt;</code>	<ul style="list-style-type: none"> <li><code>language</code></li> <li><code>name</code></li> <li><code>description</code></li> </ul>	<p>Subelement of <code>&lt;DocumentTemplateDescriptor&gt;</code> – Enter a valid <code>GwLanguage</code> value for <code>language</code>, such as <code>gw.i18n.ILocale.LANGUAGE_EN_US</code>, which uses the language typecode <code>en_US</code>. The language typecode must be defined in the <code>LanguageType</code> typelist. See also, “Obtaining an <code>ILocale</code> Object for a <code>Locale</code> Type” on page 75.</p> <p>You can also localize the name and description of this template as it appears in PolicyCenter.</p>
<code>&lt;ContextObjectLocalization&gt;</code>	<ul style="list-style-type: none"> <li><code>language</code></li> <li><code>display-name</code></li> </ul>	<p>Subelement of <code>&lt;ContextObject&gt;</code> – Enter a valid <code>GwLanguage</code> value for <code>language</code>. See the previous description for more information.</p> <p>You can also localize the name of this template as it appears PolicyCenter.</p>



To localize a document template descriptor file, add the appropriate `<DescriptorLocalization>` and `<ContextObjectLocalization>` subelements to the file.

**IMPORTANT** There is only one copy of each document template descriptor file. Do not create additional copies in locale folders. Instead, add localization elements to the descriptor files in the `doctemplates` folder.

## Localizing Email and Note Descriptor Files

To localize email and note descriptor files, you put a copy of each descriptor file in the correct locale folder and localize the following attributes in that file.

Element	Attribute	Description
<code>&lt;emailtemplate-descriptor&gt;</code> <code>&lt;notetemplate-descriptor&gt;</code>	<ul style="list-style-type: none"><li>keywords</li><li>subject</li></ul>	Localize the keywords associated with this template to facilitate the search for this template in the PolicyCenter search screen. Also, localize the subject of this template to show that localized value in PolicyCenter.

For example, to localize an email or note template descriptor file for French (France), first copy the descriptor file to an `fr_FR` folder. Then localize any keywords that you want and the subject tag for the template.

## Step 4: Localize Template Files

After copying the template content files to your locale folder, as described in “Step 2: Copy Template Content Files” on page 63, you then need to translate them. Unless you want to create a new template, the simplest procedure is to do the following:

- Open the copied base language content template.
- Translate it into the language of your choice.
- Save the translated file in the locale-specific configuration folder.


## Step 5: Localize Documents, Emails, and Notes in PolicyCenter

After you create localized versions of your templates, you can then use these templates in PolicyCenter to create a language-specific version of a document, an email, or a note.

**Note:** In PolicyCenter, you can select the unlocalized templates that are in the default directory. PolicyCenter displays these templates with no language specified. If you select one, however, PolicyCenter makes the **Language** field in the **New Document** worksheet editable.

### To create a localized document

1. In PolicyCenter, open a policy and navigate to **Actions** → **New Document** → **Create a new document from a template**. This action opens the **New Document** worksheet at the bottom of the screen.

2. In the **New Document** worksheet, click the search icon  in the **Document Template** field.

**Note:** The base configuration *Sample Acrobat* document (`SampleAcrobat.pdf`) uses Helvetica font. If you want to create a document that uses Unicode characters, such as one that uses an East Asian language, then the document template must support a Unicode font. Otherwise, the document does not display Unicode characters correctly.

3. In the search screen that opens, set the **Language** field to your language and set the other search fields as needed. If a document template for your language exists, PolicyCenter displays it in **Search Results**.
4. Click **Select**. PolicyCenter returns to the **New Document** worksheet with the selected localized template.

5. Complete the rest of the worksheet fields as necessary. You can enter text in your chosen language in the appropriate fields to further localize the document.

**To create a localized email**

1. In PolicyCenter, open a policy and choose **Actions** → **New Email** to open the **New Email** worksheet at the bottom of the screen.
2. In the **Email** worksheet, you can either enter text in your chosen language or click **Use Template** to open the template selection worksheet.
3. If you click **Use Template**:
  - a. In the search screen that opens, set the **Language** field to your chosen language and set the other search fields as necessary. If an email template for the language exists, PolicyCenter displays it in **Search Results**.
  - b. Click **Select**. PolicyCenter returns to the **New Email** worksheet with the selected localized template.
4. Complete the rest of the worksheet fields as needed. You can enter text in your chosen language in appropriate fields to further localize the document.

**To create a localized note**

1. In PolicyCenter, open a policy and choose **Actions** → **New Note** to open the **New Note** worksheet at the bottom of the screen.
2. If you click **Use Template**:
  - a. In the search screen that opens, set the **Language** field to your chosen language and set the other search fields as necessary. If an email template for the language exists, PolicyCenter displays it in **Search Results**.
  - b. Click **Select**. PolicyCenter returns to the **New Note** worksheet with the selected localized template.
3. Complete the rest of the worksheet fields as needed. You can enter text in your chosen language in appropriate fields to further localize the document.

## Document Localization Support

PolicyCenter provides a number of useful methods for working with document localization in the following APIs. You can use methods of classes that implement the interfaces described in the following topics to search for document templates and generate documents by using a specific locale:

- “IDocumentTemplateDescriptor Interface Methods” on page 67
- “IDocumentTemplateSource Plugin Interface Methods” on page 67
- “IDocumentTemplateSerializer Plugin Interface Methods” on page 67

**See also**

- “Document Creation” on page 103 in the *Rules Guide* for general information on APIs used in document creation.
- “Document Management” on page 247 in the *Integration Guide* for integration-related issues in document creation.

## IDocumentTemplateDescriptor Interface Methods

If you need to customize the PolicyCenter interface, Guidewire provides the following methods on the IDocumentTemplateDescriptor interface for working with locales:

Method	Return type	Returns
getLanguage()	String	Language to use to create the document from this template descriptor. A return value of null indicates the default language for the application.  PolicyCenter returns the language that matches the directory that contains the .descriptor file that the user selected in the PolicyCenter interface.
getName(locale)	String	Localized name from the document descriptor file for a given locale. If a translation does not exist, the method returns null.

You can also use the following property on IDocumentTemplateDescriptor to retrieve the locale:

```
gw.plugin.document.IDocumentTemplateDescriptor.locale
```

### See also

- “Localizing Document Descriptor Files” on page 64

## IDocumentTemplateSource Plugin Interface Methods

The IDocumentTemplateSource plugin provides the following useful getter methods.

Method	Return type	Returns
getDocumentTemplate(date, valuesToMatch, maxResults)	IDocumentTemplateDescriptor[]	Array of IDocumentTemplateDescriptor objects. If you need to perform a search based on the locale of a document template, you can specify this in the valuesToMatch argument.
getDocumentTemplate(templateId, locale)	IDocumentTemplateDescriptor	Document template instance corresponding to the specified template ID and locale.
getTemplateAsStream(templateId, locale)	InputStream	One of the following: <ul style="list-style-type: none"> <li>• Returns an InputStream for reading the specified template</li> <li>• Returns null if PolicyCenter cannot find the template</li> <li>• Returns null if the caller does not have adequate privileges to retrieve the template</li> </ul>

## IDocumentTemplateSerializer Plugin Interface Methods

The IDocumentTemplateSource plugin provides the following method that you can use to retrieve a localized version of a template.

Method	Return type	Returns
localize(locale, descriptor)	IDocumentTemplateDescriptor	Localized instance of the specified template



# Regional Format Configuration



# Working with Regional Formats

You can configure support for multiple regional formats in PolicyCenter. Regional formats specify how to format items like dates, times, numbers, and monetary amounts for use in the user interface. Regional formats specify the visual format of data, not the database representation of that data.

This topic includes:

- “Configuring Regional Formats” on page 71
- “Setting the Default Application Locale for Regional Formats” on page 75
- “Setting Regional Formats for a Block of Gosu Code” on page 75

## Configuring Regional Formats

In PolicyCenter, you can either use the regional formats defined in the International Components for Unicode (ICU) Library, or you can override those formats by defining formats in `localization.xml`. You save this file in the localization folder for the region it defines. If a localization folder does not have a `localization.xml` file, PolicyCenter uses the ICU Library. You configure these formatting options in Guidewire Studio.

This topic includes:

- “About the International Components for Unicode (ICU) Library” on page 71
- “Configuring Locale Codes for Default Application Locale and the ICU Library” on page 72
- “Configuring a `localization.xml` file” on page 72

### About the International Components for Unicode (ICU) Library

The International Components for Unicode (ICU) library is an open source project that provides support for Unicode and software globalization. PolicyCenter includes this library.

The ICU library, `icu4j`, attempts to maintain API compatibility with the standard Java JDK. However, for most features, the ICU library provides significant performance improvements and a richer feature set than the Java JDK. The core of the ICU library is the Common Locale Data Repository (CLDR). The CLDR repository is a 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 Locale Codes for Default Application Locale and the ICU Library

If a localization folder does not contain a `localization.xml` file, PolicyCenter uses the ICU library defaults for that region, as follows:

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

ICU uses Java locale codes to identify specific regional settings. For PolicyCenter to be able to access the ICU library for a region, the locale code for that region must be defined in the `LocaleType` typelist. In the base configuration, PolicyCenter provides the following set of Java locale codes in the `LocaleType` typelist:

```
en_US United States (English)
en_GB Great Britain (English)
en_CA Canada (English)
en_AU Australia (English)
fr_CA Canada (French)
fr_FR France (French)
de_DE Germany (German)
ja_JP Japan (Japanese)
```

The Java locale codes defined in this typelist are used by PolicyCenter as follows:

- To define the locale codes that you can use to set the default application locale. See “Configuring Locale Codes for Default Application Locale and the ICU Library” on page 72.
- To access the regional formats for a locale supplied by the ICU library. See “About the International Components for Unicode (ICU) Library” on page 71.

You can add additional Java locale codes to the `LocaleType` typelist.

### To add a Java locale code to the `LocaleType` typelist

1. In PolicyCenter Studio, navigate in the **Project** window to `configuration` → `config` → `Metadata` → `Typelist`.
2. Right-click `LocaleType.tti` and choose **New** → **Typelist Extension**.
3. Click OK in the dialog box to create `LocaleType.ttx` in the `extensions/typelist` folder.

The Typelist editor opens so you can edit the new `LocaleType.ttx` file.

**Note:** You need to perform these steps only the first time you add a new Java locale code. To add subsequent locale codes, just navigate to `configuration` → `config` → `Extensions` → `Typelist` and double-click `LocaleType.ttx`.

4. Right-click the typelist element and choose **Add new** → **typecode**.
5. For **code**, enter the Java locale code.  
For example, enter the Java locale code `n1_NL` to configure regional formats used in the Netherlands.
6. For **name** and **description**, by convention, specify the country followed by the language in parentheses.  
For example, enter `Netherlands (Dutch)` for both the name and the description.

## Configuring a `localization.xml` file

A localization folder can contain a `localization.xml` file, which defines regional formats. If you open Guidewire Studio and navigate in the **Project** window to `configuration` → `config` → `Localizations`, you can see the local-



ization folders provided in the base configuration of PolicyCenter. These localization folders have Java locale codes for names, such as:

- de\_DE
- en\_US
- fr\_FR
- ja\_JP

In the base configuration, the en\_US, fr\_FR, and ja\_JP localization folders have `localization.xml` files that define regional formats customary to each region. 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:** In the base configuration, Guidewire uses the ICU library defaults for certain regions, such as de\_DE. For those regions, there is no `localization.xml` file. See “International Components for Unicode (ICU) Library” on page 22.

You can add a new localization folder to **Localizations**. To do so, use the contextual right-click menu to create a new folder that uses a Java locale code defined in the `LocaleType` typelist. If there are multiple copies of a localization folder, then the contents of the folder in the main **configuration** module override any files installed by a language pack.

You can use an existing `localization.xml` file as a starting point if you want to create a new `localization.xml` file to override the ICU library settings for a region. You define attributes and subelements of the `<GWLlocale>` element in a `localization.xml` file, as described in the following topic.

## <GWLlocale> XML Element

A `localization.xml` file contains a single `<GWLlocale>` element in which you define the settings for a region. If you do not define an element or attribute, PolicyCenter uses the ICU library setting. The `<GWLlocale>` element can take the following attributes and subelements.

### <GWLlocale> Attributes

- **code** – The region identifier, typically the same as the Java locale code defined in `LocaleType`. For example, "en\_US".
- **name** – A String value for the name of the locale, which the application can display to the user in its screens. For example, "United States (English)".
- **firstDayOfWeek** – Defines the first day of the week for the region. Value is an integer representing a day of the week, starting with "1" for Sunday, "2" for Monday, and so on.

If not defined, the base configuration uses the default ICU library setting for the region:

- **Sunday** – en\_AU, en\_CA, en\_US, fr\_CA, ja\_JP
- **Monday** – de\_DE, en\_GB, fr\_FR
- **typecode** – The corresponding regional typecode defined in the `LocaleType` typelist. For example, "en\_US".
- **defaultCalendar** – "Gregorian" or "JapaneseImperial". The default value is "Gregorian".
- **enableJapaneseCalendar** – A Boolean value, true or false, that determines whether or not to enable the Japanese calendar for this region. See “Working with the Japanese Imperial Calendar” on page 77.

### <GWLlocale> Subelements

- **CurrencyFormat** – Currency format pattern for the region, used in single currency rendering mode. For multicurrency configuration information, see “PolicyCenter Base Configuration Currencies” on page 87.

Attributes are:

- `negativePattern` – Pattern for a negative currency value. For example, for U.S. dollars, "\$#").
- `positivePattern` – Pattern for a positive currency value. For example, for U.S. dollars, "\$#".
- `zeroValue` – Pattern for a scalar value of zero. For example, "-".
- `DateFormat` – Patterns for the date formatter and parser. Attributes are:
  - `long` – The long date format pattern. For example, "E, MMM d, yyyy".
  - `medium` – The medium date format pattern. For example, "MMM d, yyyy".
  - `short` – The short date format pattern. For example, "MM/dd/yyyy".
 See <http://cldr.unicode.org/translation/date-time-patterns>.
- `FlexibleDateFormat` – Date format patterns in addition to short, medium, and long. This element is used only by ClaimCenter. Attributes are:
  - `year-month-day` – Format that contains year, month, and day. For example, "M/d/y" or "y-MM-dd".
  - `year-month` – Format that contains year and month. For example, "MMM y" or "y MMM".
  - `month-day` – Format that contains month and day. For example, "MMM d" or "MM-dd".
- `JapaneseImperialDateFormat` – Japanese imperial calendar settings. See “Working with the Japanese Imperial Calendar” on page 77.
 Attributes are:
  - `long` – Long Japanese imperial date format.
  - `medium` – Medium Japanese imperial date format.
  - `short` – Short Japanese imperial date format.
  - `yearSymbol` – Japanese year symbol.
- `NumberFormat` – Number formatter and parser configurations. Attributes are:
  - `decimalSymbol` – Symbol used to denote a decimal, usually a period or a comma. For example, ".".
  - `negativeEntryPattern` – Format for entering negative numeric values. For example, "(#)".
  - `thousandsSymbol` – Symbol used to denote a grouping separator, usually a period or a comma. For example, ",".
- `TimeFormat` – Time formatter and parser configuration. Attributes are:
  - `long` – Long time format pattern. For example, "h:mm:ss a z".
  - `medium` – Medium time format pattern. For example, "hh:mm:ss a".
  - `short` – Short time format pattern. For example, "h:mm a".
 See <http://cldr.unicode.org/translation/date-time-patterns>.
- `NameFormat` – Formatting of names by the PCF file `GlobalContactNameInputSet` or `GlobalPersonNameInputSet`. Attributes are:
  - `PCFMode` – Mode of the PCF file to use. The mode must exist. In the base configuration, there are two modes, `default` and `Japan`. The default value in the base configuration is `default`.
  - `textFormatMode` – How to format the text. In the base configuration, `Japan` and `France` are possible values.
  - `visibleFields` – Fields the user can see in the PCF file `GlobalContactNameInputSet` or `GlobalPersonNameInputSet`. For example, for `France`, the visible fields are "Prefix, FirstName, MiddleName, Particle, LastName, Suffix, Name".
- `CalendarWidget` – Defines the year-month pattern used by the calendar widget. Attribute is:
  - `yearMonth` – For example, for `ja_JP`, the value is "yyyyMM".

## 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 PolicyCenter with a single region as the choice for regional formats. You set the default application locale in Guidewire Studio by editing the configuration parameter `DefaultApplicationLocale` in the file `config.xml`. In the base configuration, the default application locale is set to `en_US`.

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

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

---

**IMPORTANT** 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.

---

### See also

- “Configuring Locale Codes for Default Application Locale and the ICU Library” on page 72

## Setting Regional Formats for a Block of Gosu Code

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

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

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

The second method sets both region and language at the same time.

Running a block of Gosu code in this way enables PolicyCenter to format dates, times, numbers, and names of people appropriately for a specific region. Otherwise, the block of Gosu code uses the regional formats specified by the current region. The current region is either specified by the current user or, if not, by the `DefaultApplicationLocale` parameter in `config.xml`.

The parameters that the methods can take are:

Parameter	Description
<code>alternateLocale</code>	An object of type <code>ILocale</code> that represents a regional format from the <code>LocaleType</code> typelist. Specify a <code>GWLocale</code> object for this parameter.
<code>alternateLanguage</code>	An object of type <code>ILocale</code> that represents a language from the <code>LanguageType</code> typelist. . Specify a <code>GWLanguage</code> object for this parameter.
<code>\ -&gt; { <i>code</i> }</code>	A Gosu block as a <code>GWRunnable</code> object—the Gosu code to run with different regional formats or a different language

### Obtaining an ILocale Object for a Locale Type

To run a block of Gosu code with a specified set of regional formats, you must specify a locale object of type `ILocale`. You must specify the subtype `GWLocale` or `GWLanguage` as appropriate for the parameter `alternateLocale` or `alternateLanguage`.

- For the parameter `alternateLocale`, use the `gw.api.util.LocaleUtil.toLocale` method to provide an `ILocale` object that corresponds to a Gosu typecode in the `LocaleType` typelist. The object is actually of type

`GWLocale`, which implements `ILocale`. You can specify the object directly by using typecode syntax. For example `LocaleType.TC_EN_US`. The typecode has to be defined in the `LocaleType` typelist.

- For the parameter `alternateLanguage`, use the `gw.api.util.LocaleUtil.toLanguage` method to provide an `ILocale` object that corresponds to a Gosu typecode in the `LanguageType` typelist. The object is actually of type `GWLanguage`, which implements `ILocale`. You can specify the object directly by using typecode syntax. For example `LanguageType.TC_EN_US`. The typecode has to be defined in the `LanguageType` typelist.
- You can specify a typecode of the correct type directly, without using the `toLocale` or `toLanguage` method. Use the following syntax:
  - `GWLocale – gw.i18n.ILocale.FR_FR`
  - `GWLanguage – gw.i18n.ILocale.LANGUAGE_FR_FR`
- You can specify the first parameter by using a method on `LocaleUtil` that can get the current or default locale or language, as appropriate. For example, `getDefaultLocale` or `getDefaultLanguage`.

You can add a typecode to the `LocaleType` or `LanguageType` typelist. If you add a new typecode to one of these typelists, you must restart Studio to be able to use it in `gw.i18n.ILocale`. For example, see “To add a Java locale code to the `LocaleType` typelist” on page 72.

The following example Gosu code formats today’s date by using the default application regional formats, overriding the any region that the user might have specified. The code uses `runAsCurrentLocale` to run a block of code that prints today’s date formatted according the default application region’s date format. The first parameter to this method is a call to `getDefaultLocale` to obtain a `GWLocale` object that represents the application’s default regional formats.

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

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

#### See also

- “Setting a Language for a Block of Gosu Code” on page 51
- “Gosu Blocks” on page 231 in the *Gosu Reference Guide*

# Working with the Japanese Imperial Calendar

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

This topic includes:

- “The Japanese Imperial Calendar Date Widget” on page 77
- “Configuring Japanese Dates” on page 78
- “Setting the Japanese Imperial Calendar as the Default for a Region” on page 79
- “Setting a Field to Always Display the Japanese Imperial Calendar” on page 80
- “Setting a Field to Conditionally Display the Japanese Imperial Calendar” on page 80
- “Working with Japanese Imperial Dates in Gosu” on page 82

## The Japanese Imperial Calendar Date Widget

The PolicyCenter user 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:

日付 / 時間: 09:41

平成 22年 9月

平成 昭和 大正 明治

火 水 木 金 土

1 2 3 4

7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30

今日 承認

平成 22/09/02 09:41

The date widget displays only 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 by using 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 PolicyCenter display of Japanese Imperial calendar dates by specifying the `<JapaneseImperialDateFormat>` element in `localization.xml`. The `<JapaneseImperialDateFormat>` element is similar to the existing Gregorian `<DateFormat>` element. You define the following attributes:

Date format	Description
long medium	PolicyCenter uses only the long and medium date formats to display Japanese dates. You cannot use these fields to format user input data.
short	PolicyCenter 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 <code>short="G yy/MM/dd"</code> .
yearSymbol	PolicyCenter uses the Japanese yearSymbol as a signal to render the Japanese Imperial calendar date picker.

The following graphic shows the base configuration Japanese <GWLocale> definition, which is in localization.xml in the localization folder ja\_JP. This definition includes the <JapaneseImperialDateFormat> element.



## Setting the Japanese Imperial Calendar as the Default for a Region

PolicyCenter provides a way to set a default calendar for a region through the defaultCalendar attribute of the <GWLocale> element. Additionally, for the Japanese Imperial calendar, you must also set the enableJapaneseCalendar of the <GWLocale> element to true. These attributes determine whether or not to enable the Japanese calendar for this region.

**To set the Japanese Imperial calendar as the default calendar for Japan:**

1. Open the file configuration → config → ja\_JP → localization.xml.
2. Add the defaultCalendar attribute to the <GWLocale> element, with the value set to JapaneseImperial.
3. Add the enableJapaneseCalendar attribute to the <GWLocale> element, with the value set to true.

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

**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, require 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

- “Working with Regional Formats” on page 71.

## Setting a Field to Always Display the Japanese Imperial Calendar

You can set a field of an entity to always display the Japanese Imperial calendar when it is used the PolicyCenter user interface.

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

1. Navigate in the Studio Project window to **configuration** → **config** → **Extensions** → **Entity**, and then double-click `Activity.etx` to open it in the editor.
2. Right-click an element on the left, such as **entity (extension)** and choose **Add new** → **column**.
3. Select the new column and enter the following values for the following attributes:
  - **name** – `JICDate`
  - **type** – `japaneseimperialdate`
  - **nullok** – `true`

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:

```

Screen : ActivitiesScreen
Title Bar
Activities
Toolbar
List View : ActivitiesLV
Row Iterator activity
  Subject Create Date Due Date Japanese Date
Row
  activity.Subject activity.CreateTime activity.TargetDate activity.JICDate
  
```

**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 PolicyCenter to display certain fields in different formats in different situations. For example (as described in



“Working with Japanese Imperial Dates in Gosu” on page 82), suppose that you want to do the following:

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

PolicyCenter 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, add a column with the following attributes:

- `name` – `JICDate`
- `type` – `japaneseimperialdate`
- `nulloK` – `true`

See “Setting a Field to Always Display the Japanese Imperial Calendar” on page 80.

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, `IDataTypeDef`), then 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 prompts you with the required method and property bodies to automatically insert. 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 239 in the *Configuration Guide*. For sample code for a JIC presentation handler, see “Working with Japanese Imperial Dates in Gosu” on page 82.

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” on page 81.
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” on page 81.

#### See Also

- For an example of how to implement the `IDataTypeDef` interface, see “Defining a New Tax Identification Number Data Type” on page 239 in the *Configuration Guide*. Specifically, see “Step 2: Implement the `IDataTypeDef` Interface” on page 240 in the *Configuration Guide*.
- “Working with Japanese Imperial Dates in Gosu” on page 82

## Working with Japanese Imperial Dates in Gosu

PolicyCenter 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 option 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 <code>Policy.PolicyType</code> is null (unspecified), then PolicyCenter displays all date fields in the Gregorian calendar in all situations
ja_JP	–	If <code>Policy.PolicyType</code> is null (unspecified), then PolicyCenter displays all date fields in the Japanese Imperial calendar in all situations
en_US	=="CALI"	If <code>Policy.PolicyType</code> = "CALI", then PolicyCenter displays <code>Policy.EffectiveDate</code> and <code>Policy.ExpirationDate</code> in the Japanese Imperial calendar and all other date fields in the Gregorian calendar.
ja_JP	!="CALI"	If <code>Policy.PolicyType</code> != "CALI", then PolicyCenter displays <code>Policy.EffectiveDate</code> and <code>Policy.ExpirationDate</code> 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 auto liability coverage in the United States. A driver who wants personal coverage must purchase another policy, known as a Voluntary policy.

These cases result in the following behavior:

- 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, PolicyCenter 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 80.

#### See Also

- “Setting the Japanese Imperial Calendar as the Default for a Region” on page 79

- “Setting a Field to Always Display the Japanese Imperial Calendar” on page 80
- “Setting a Field to Conditionally Display the Japanese Imperial Calendar” on page 80



# National Formats and Data Configuration



# Configuring Currencies

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

This topic includes:

- “PolicyCenter Base Configuration Currencies” on page 87
- “Monetary Amounts in the Data Model and in Gosu” on page 89
- “Currency-related Configuration Parameters” on page 92

## PolicyCenter 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 whether you configure your installation for single currency or multicurrency mode:

Studio	Single currency rendering mode	Multicurrency rendering mode
Currency typelist	Provides currency code and similar information for the supported currency. <ul style="list-style-type: none"> <li>In a new installation, remove all currency typecodes except that of the default application currency.</li> <li>In an installation that you are upgrading, retire all Currency typecodes except that of the default application currency.</li> </ul>	Contains information on all supported currencies, such as currency codes and similar information.
currency.xml	In single currency mode, do not use this file to define currency formats. Remove all currency formatting from this file. Instead, use file <code>localization.xml</code> to format currency information.	Studio supports multiple copies of <code>currency.xml</code> , one for each supported currency, in each currency folder. For example: configuration → config → currencies → aud → <code>currency.xml</code>
datatypes.xml	Use to set the following for the application currency—the default application currency: <ul style="list-style-type: none"> <li>precision</li> <li>scale</li> <li>appscale</li> </ul>	Use to set the following for the default currency only: <ul style="list-style-type: none"> <li>precision</li> <li>scale</li> <li>appscale</li> </ul> Use individual <code>currency.xml</code> files to set the <code>storageScale</code> attribute on the <code>&lt;CurrencyType&gt;</code> element for each defined currency.
localization.xml	Contains currency formatting information for use with single currency rendering mode only. Studio supports multiple copies of this file, one for each localization folder. For example: configuration → config → Localizations → en_US	In multicurrency mode, do not use this file to define currency formats. Remove all currency formatting from <code>localization.xml</code> .

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` to be the same in each application.

#### See also

- “Setting the Default Application Currency” on page 92
- “Choosing a Rounding Mode” on page 92
- “Setting the Currency Display Mode” on page 93

## Working with Currency Typecodes

The typecodes in the Currency typelist determine which currencies you can use to enter monetary amounts in PolicyCenter. 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.ttx typelist either by:
  - Searching for it with Ctrl+Shift+N.
  - Navigating in the Project window to **configuration** → **config** → **Extensions** → **Typelist** and double-clicking Currency.ttx.
2. In the toolbar, click the Add icon .
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 PolicyCenter.

In the base configuration, Guidewire defines the <monetaryamount> subelement in metadata definition files for fields that handle monetary amounts. Use the monetary amount data type to store and display monetary amounts within Guidewire PolicyCenter.

The <monetaryamount> subelement is a compound data type that stores its values in two separate database columns. One of the columns is <money> column type, and the other is a typekey field to the currency typelist.

**IMPORTANT** Do not use the currencyamount column type in your data model. Do not use the corresponding CurrencyAmount Gosu or Java type. Use the <monetaryamount> subelement instead.

## Monetary Amount Data Type

The money data types store and show monetary amounts in the system. PolicyCenter assumes that 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`

These settings that you make for these parameters affect all money columns, which are a component of the `<monetaryamount>` subelement.

### 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 enables you to later add currencies for which the number of decimal digits is 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 PolicyCenter displays monetary amounts in the user interface and how PolicyCenter stores monetary amounts in the database.

#### See also

- “The Data Types Configuration File” on page 234 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 PolicyCenter saves to and loads from monetary columns in the database.

For example, suppose that you implement a single currency configuration of PolicyCenter 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 `monetaryamount` columns have a scale of zero but are stored in the database with a scale of two. Later, during your conversion to a configuration with multiple currencies, 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 each currency.

Otherwise, you must re-create 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 PolicyCenter 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-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...
<code>DefaultApplicationCurrency</code>	Default currency for the application	"Setting the Default Application Currency" on page 92
<code>DefaultRoundingMode</code>	Default rounding mode for money and currency amount calculations	"Choosing a Rounding Mode" on page 92
<code>MulticurrencyDisplayMode</code>	Determines whether PolicyCenter displays currency selectors for monetary values	"Setting the Currency Display Mode" on page 93

---

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

---

### See also

- "Globalization Parameters" on page 56 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 PolicyCenter with a single currency.

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

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

Guidewire applications that share currency values must have the same `DefaultApplicationCurrency` setting in their respective `config.xml` files.

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

---

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

---

## Choosing a Rounding Mode

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

By default, PolicyCenter 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 58 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 PolicyCenter. You must set a currency display mode for PolicyCenter. Set the parameter to one of the following values:

- SINGLE
- MULTIPLE

Setting the currency display mode to SINGLE disables currency selectors for entering monetary amounts in the user interface. If you configure PolicyCenter with one currency only, then set `MulticurrencyDisplayMode` to SINGLE. If you configure PolicyCenter with multiple currencies, then set `MulticurrencyDisplayMode` to MULTIPLE.

**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 where you specify certain configuration settings that influence the display and storage of monetary amounts.

Currency display mode	Decimal digits to show...
SINGLE	<p>PolicyCenter uses the value of the <code>appscale</code> attribute on the money data type in <code>datatypes.xml</code> to determine the number of decimal digits to show in the interface. If you do not set a value for <code>appscale</code>, then PolicyCenter uses the <code>scale</code> value to determine the number of decimal digits to show.</p> <p><b>Note</b> The <code>appscale</code> attribute does not exist in the base configuration. You need to add this attribute to the <code>&lt;MoneyDataType&gt;</code> element in the <code>datatypes.xml</code> file.</p>
MULTIPLE	<p>PolicyCenter uses the value of the <code>storageScale</code> attribute on <code>&lt;CurrencyType&gt;</code> in the individual <code>currency.xml</code> files to determine the number of decimal digits to show in the interface for that currency.</p> <p>The <code>appscale</code> attribute is useful in multicurrency configurations for the money data types. However, with <code>currencyamount</code> data types, the <code>storageScale</code> attribute supersedes the <code>appscale</code> attribute on the <code>&lt;MoneyDataType&gt;</code> element in <code>datatypes.xml</code>. Make the currency <code>storageScale</code> value less than or equal to the <code>appscale</code> value and make the <code>appscale</code> value less than or equal to the <code>scale</code> value. The following must always be true.</p> <p><code>storageScale &lt;= appScale &lt;= scale</code></p>



# 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 PolicyCenter.

This topic includes:

- “Working with Country Typecodes” on page 95
- “Configuring Country Address Formats” on page 95
- “Setting the Default Application Country” on page 97
- “Configuring Jurisdiction Information” on page 97
- “Configuring Zone Information” on page 97

## Working with Country Typecodes

In the base configuration, Guidewire provides the country typelist *Country.ttx*, which 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](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 *Country.ttx* typecode definition for VA, Vatican City, indicates that the currency in use is the euro, code *eur*.

## Configuring Country Address Formats

The *country.xml* files in the country folders define settings that control the appearance of address information for each country in PolicyCenter. In Guidewire Studio, to access a country’s *country.xml* file, you can navigate in the **Project** window to **configuration** → **config** → **geodata** → *countryCode*. For example, the *country.xml* file for the address-related information in Australia is 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>`.

**Note:** For attributes that take display key values, PolicyCenter uses the `display.properties` file defined in the associated localization folder, such as `Localization → ja_JP` for Japan. If there is no `display.properties` file in the localization folder for that region, PolicyCenter uses the `display.properties` file defined in `Localization → en_US`. See “Localizing Display Keys” on page 41.

Attributes	Sets...
<code>PCFMode</code>	<p>PCF mode for <code>GlobalAddressInputSet.pcf</code>. In the base configuration, Guidewire provides three modes for showing address information. The base configuration modes are:</p> <ul style="list-style-type: none"> <li>• <code>BigToSmall</code> – Used to format addresses for Japan</li> <li>• <code>PostCodeBeforeCity</code> – Used to format addresses for France and Germany</li> <li>• <code>default</code> – Used to format all other addresses</li> </ul> <p>The default value is <code>default</code>.</p> <p>You can add new PCF modes and use them in file <code>country.xml</code>.</p>
<code>postalCodeDisplayKey</code>	<p>Name of the display key to use for the postal code label in PolicyCenter. For example:</p> <ul style="list-style-type: none"> <li>• In the United States, US – <code>Web.AddressBook.AddressInputSet.ZIP</code></li> <li>• In Japan, JP – <code>Web.AddressBook.AddressInputSet.Postcode</code></li> </ul> <p>The default value is <code>Web.AddressBook.AddressInputSet.PostalCode</code>.</p>
<code>cityDisplayKey</code>	<p>Name of the display key to use for the city label for a country. For example:</p> <ul style="list-style-type: none"> <li>• In Great Britain, GB, – <code>Web.AddressBook.AddressInputSet.TownCity</code></li> </ul> <p>The default value is <code>Web.AddressBook.AddressInputSet.City</code>.</p>
<code>stateDisplayKey</code>	<p>Name of the display key to use for the label designating the major administrative subdivisions in a country. For example:</p> <ul style="list-style-type: none"> <li>• In the United States, US – <code>Web.AddressBook.AddressInputSet.State</code></li> <li>• In Canada, CA – <code>Web.AddressBook.AddressInputSet.Province</code></li> <li>• In Japan, JP – <code>Web.AddressBook.AddressInputSet.Prefecture</code></li> </ul> <p>The default value is <code>Web.AddressBook.AddressInputSet.State</code>.</p>
<code>visibleFields</code>	<p>Comma-separated list of address-related fields that you want to be visible in PolicyCenter for this country. For example, for AU, the values are:</p> <ul style="list-style-type: none"> <li>• <code>Country</code></li> <li>• <code>AddressLine1</code></li> <li>• <code>AddressLine2</code></li> <li>• <code>AddressLine3</code></li> <li>• <code>City</code></li> <li>• <code>State</code></li> <li>• <code>PostalCode</code></li> </ul> <p>Any address field that you designate as visible in this file must correspond to the constants defined in <code>AddressOwnerFieldId.gs</code>.</p>

#### See also

- “Configuring Address Data and Field Order for a Country” on page 107
- “Address Modes in Page Configuration” on page 110



## 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](http://www.iso.org/iso/english_country_names_and_code_elements)

See “Globalization Parameters” on page 56 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, such as Berlin, Germany

In the base configuration, Guidewire provides a `Jurisdiction` typelist that contains a set of pre-defined jurisdictions. This typelist is used by a number of PCF files to display a list of states or provinces, as well as jurisdictions that are not states or provinces.

Many PolicyCenter fields that appear to reference a state actually reference a jurisdiction instead. For example:

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

## Configuring Zone Information

Guidewire PolicyCenter uses `zone-config.xml` files to define one or more zones. A *zone* is a combination of a country and zero or more address elements, such as a city or state in the United States or a province in Canada. You can configure zones to apply to any area in a single country. In the United States, for example, you typically define zones by state, city, county, and ZIP code. There is a separate `zone-config.xml` file for each country defined in the base configuration.

PolicyCenter uses zones for the following:

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

This topic includes:

- “Location of Zone Configuration Files” on page 97
- “Working with Zone Configuration Files” on page 98

### Location of Zone Configuration Files

PolicyCenter stores the `zone-config.xml` files in the `geodata` folder, each in its own individual country folder. For example, the `zone-config.xml` file for the address-related information in Australia is in the following location in Studio:

`configuration → config → geodata → AU`

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 – United States of America

In a `zone-config.xml` file, you define:

- The links between the country's zones, the *zone hierarchy*
- How PolicyCenter uses those links to extract the value of zone from address data

## Working with Zone Configuration Files

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

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

For example, in the base configuration, PolicyCenter defines the zone hierarchy for the United States in the following file:

`configuration → config → geodata → US → zone-config.xml`

This file uses the following elements:

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

  <Zone code="state" fileColumn="2" granularity="4" regionMatchOrder="3">
    <AddressZoneValue>
      Address.State.Abbreviation.DisplayName
    </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.Abbreviation.DisplayName + ":" + Address.City
    </AddressZoneValue>
    <Links>
      <Link toZone="zip"/>
    </Links>
  </Zone>

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

```
</Zone>
</Zones>
```

This topic includes:

- “<Zones>” on page 99
- “<Zone>” on page 99
- “<ZoneCode>” on page 100
- “<Links>” on page 100
- “<AddressZoneValue>” on page 101

#### See also

- “Zone Import Command” on page 163 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	<i>Required.</i> Defines the country to which this zone configuration applies.
dataFile	For Guidewire internal use only. Do not set this attribute.

### <Zone>

The <Zone> subelement of <Zones> 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, which 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	<p><i>Optional.</i> 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.</p> <p>A &lt;Zone&gt; element without a fileColumn attribute can contain 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 &lt;Zones&gt; element:</p> <pre>&lt;Zone code="fsa" regionMatchOrder="1" granularity="1"&gt;   &lt;ZoneCode&gt; postalcode.substring(0,3) &lt;/ZoneCode&gt;   &lt;AddressZoneValue&gt; Address.PostalCode.substring(0,3) &lt;/AddressZoneValue&gt; &lt;/Zone&gt;</pre> <p>Both the &lt;ZoneCode&gt; and &lt;AddressZoneValue&gt; elements extract data from the actual address data by parsing the data into substrings.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• Specify at least one &lt;Zone&gt; element with a fileColumn attribute. If you do not specify at least one fileColumn attribute, then PolicyCenter does not import data from the address zone data file.</li> <li>• Import address zone data upon first installing Guidewire PolicyCenter, and then at infrequent intervals thereafter as you update zone definitions.</li> </ul>
granularity	Sets size levels 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. PolicyCenter uses this value with holidays and business weeks.
orgZone	For Guidewire internal use only. Do not set this attribute.

Attribute	Description
regionMatchOrder	<p>Controls the order in which PolicyCenter uses these zones in matching algorithms. PolicyCenter uses this attribute as it matches users to a zone for location-based or proximity-based assignment. For example, in the base configuration for the United States, Guidewire defines the following &lt;Zone&gt; attributes for a county:</p> <pre>&lt;Zone   code="county"   fileColumn="4"   regionMatchOrder="2"   granularity="3" &gt;   ... &lt;/Zone&gt;</pre> <p>Setting the regionMatchOrder to 2 means that PolicyCenter matches county data second, after another zone, while matching a user to a location.</p> <p>The county value also:</p> <ul style="list-style-type: none"> <li>• Appears in the fourth column of the data file.</li> <li>• Is third in granularity, one size less than a state—granularity 4—and one size more than a city—granularity 2.</li> </ul>
unique	<p><i>Optional.</i> 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.</p> <p>To differentiate zones that can be identical—not unique—use a &lt;ZoneCode&gt; 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 by defining the following &lt;ZoneCode&gt; subelement of &lt;Zone&gt;:</p> <pre>&lt;ZoneCode&gt;   state + ":" + county &lt;/ZoneCode&gt;</pre> <p>See “&lt;ZoneCode&gt;” on page 100.</p>

### <ZoneCode>

The <ZoneCode> element is a subelement of <Zone>. It is possible for zone information to not be unique, meaning that the zone import data column contains two or more identical values. In this case you need to use <ZoneCode> to define an expression that uniquely identifies the individual zone.

For example, in the United States, it is possible for multiple states to have a city with the same name, such as Portland, Oregon and Portland, Maine. To uniquely identify the city, you associate a particular state with the city. To make this association, 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:

```
<Zone code="city" fileColumn="3" granularity="2">
  <ZoneCode>
    state + ":" + city
  </ZoneCode>
  ...
</Zone>
```

This expression instructs PolicyCenter to concatenate the State value with the County value, separated by a colon delimiter. The values you use to construct the expression must be valid <Zone> code values, other than constants, that are defined in a <Zones> element for this country.

#### See also

- “<Zone>” on page 99

### <Links>

The <Links> element is a subelement of <Zone>. 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 has the following attributes:

Attributes	Description
lookupOrder	<i>Optional.</i> Tells the application the order in which to apply this value while performing a lookup. Specifying a lookup order can increase performance somewhat. If you do not specify a <code>lookupOrder</code> value, PolicyCenter uses the order that appears in the file.
toZone	<i>Required.</i> Defines a relationship to another zone for PolicyCenter to use if the <code>&lt;Zone&gt;</code> code value is not available for lookup.

For example, the following code defines a relationship between one zone type, county, and several other zone types. PolicyCenter uses these other 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 PolicyCenter cannot look up the address by its county value, then PolicyCenter attempts to look up the address first by zip value.
- The second `<Link>` definition, `<Link toZone="city" lookupOrder="2"/>`, creates a link from county to city. If PolicyCenter cannot look up the address by its county value or its zip value, it looks up the address by its city value.

#### See also

- “`<Zone>`” on page 99

### `<AddressZoneValue>`

The `<AddressZoneValue>` element is a subelement of `<Zone>`. 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.

PolicyCenter 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, PolicyCenter defines a `<Zone>` element of type `postalcode` for Canada. It looks similar to the following:

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

Given this definition, PolicyCenter 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.

#### See also

- “`<Zone>`” on page 99



# Configuring Address Information

Address data and formats in Guidewire PolicyCenter vary by country. For example, PolicyCenter shows a ZIP code or postal code field for an address depending on the country of the address. You can customize this feature, as described in this topic.

This topic includes:

- “Addresses in PolicyCenter” on page 103
- “Understanding Global Addresses” on page 104
- “Configuring Address Data and Field Order for a Country” on page 107
- “Address Modes in Page Configuration” on page 110
- “Address Owners” on page 111
- “AddressOwnerFieldId Class” on page 111
- “Automatic Address Completion and Fill-in” on page 112

## Addresses in PolicyCenter

In PolicyCenter, an address represents a street or mailing address. An address can also contain geographic location information for use in proximity searches.

Guidewire PolicyCenter uses address information for the following business entities:

- Contacts
- Account locations
- Policy locations
- Policy addresses
- Producer addresses

PolicyCenter bases Organizations and Users on Contacts. Therefore, these objects use addresses as well. PolicyCenter also uses address-related information for the following:

- Territory codes

- Tax locations
- WC excluded workplaces

However, these last items are largely specific to the United States and you can generally ignore them if you are not using the en\_US region.

### Read-only and Editable Addresses

PolicyCenter displays address information as read-only text or as editable text entry fields:

- PolicyCenter displays read-only addresses as read-only text on multiple lines. PolicyCenter uses the `country.xml` file for the current country and the `AddressFormatter` class to determine which address fields to show.
- PolicyCenter displays editable addresses as a set of editable text fields in which you can add, modify, or delete information. PolicyCenter uses the `country.xml` file for the current country to determine the address fields to show.

PolicyCenter displays a range selector—a drop-down list—in the address screen to enable you to select from an array of appropriate addresses. If you cannot edit an address, the address fields are dimmed. The drop-down list can also display a **New...** command that you can use to create a new address.

### Address Owners

PolicyCenter builds the address view around the concept of an address owner. The address owner identifies the object that owns a particular address. Additionally, the address owner controls how the address widget looks in the user interface and ensures that PolicyCenter saves the address properly. You can define different address owners depending on your requirements.

### See also

- “Understanding Global Addresses” on page 104
- “AddressFormatter Class” on page 105
- “Address Modes in Page Configuration” on page 110
- “Address Owners” on page 111
- “AddressOwnerFieldId Class” on page 111

## Understanding Global Addresses

The information that you see for an address depends on:

- The country of the address.
- Whether PolicyCenter displays the address as text entry fields or read-only text, as described at “Read-only and Editable Addresses” on page 104.

This topic includes:

- “Country XML Files” on page 105
- “Modal Address PCF Files” on page 105
- “AddressFormatter Class” on page 105
- “Addresses and States or Jurisdictions” on page 107
- “Address Configuration Files” on page 107



## Country XML Files

The `country.xml` files define settings that control the appearance of address information in PolicyCenter. In Guidewire Studio, you can see that there is a `country.xml` file for each defined country. PolicyCenter 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 is in the following location in Studio:

`configuration → config → geodata → AU`

See “Configuring Address Data and Field Order for a Country” on page 107.

## Modal Address PCF Files

In general, the country of an address determines which address fields in the database are visible in the PolicyCenter 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 PolicyCenter displays address fields.

You can configure the modal versions of PCF `GlobalAddressInputSet` to provide a modal version for each country for which you want to support address editing. 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, such as prefecture in Japan, state in the United States, and province in Canada.

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

- `GlobalAddressInputSet.default` – Used for Japan.
- `GlobalAddressInputSet.BigToSmall` – Used for Australia, Canada, Great Britain, and the United States.
- `GlobalAddressInputSet.PostCodeBeforeCity` – Used for France and Germany.

To determine which modal PCF file is used for a country, you set the `PCFmode` attribute in the `country.xml` file for that country. See “PCFMode Attribute of the Country XML File” on page 108.

### See also

- For more information on `GlobalAddressInputSet`, see “Address Modes in Page Configuration” on page 110.
- For more information on modal PCF files, see “Working with Shared or Included Files” on page 289 in the *Configuration Guide*.

## AddressFormatter Class

The `AddressFormatter` class is used to handle address display for globalization. If you change, add, or delete columns of the `Address` entity, you must also update this class. Additionally, if you add a new country definition, you might need to update the switch statement in the `internalFormat` method of this class.

There are two types of addresses in PolicyCenter:

- Synchronized
- Unsynchronized

Unsynchronized addresses apply to policy location addresses and policy addresses. An address is unsynchronized if the following are all true:

- The address is part of a completed job (a promoted branch).
- The associated account location address has changed since the job was completed.

Unsynchronized addresses are read-only. There are several ways to handle this situation:

- Expand the unsynchronized address definition in the PCF to apply to the entire address. Currently, PolicyCenter does not treat the street address portion of the PCF as either synchronized or unsynchronized.

- Format the display string using the `gw.address.AddressFormatter` class.

## Displaying Read-only Address Information

Guidewire PolicyCenter displays read-only address information in several different places. PolicyCenter displays read-only address information as a string in which the address values are separated by commas or by line-feeds. PolicyCenter uses class `gw.address.AddressFormatter` to manage and format read-only strings.

The `AddressFormatter` class consists of two parts:

- The class contains variables for all the address columns. You can extend the class to add new variables if you extend the `Address` entity with new columns.
- The class contains two versions of the `format` method with different signatures.
  - The following version of the `format` method formats an address as text and by default includes all fields. The `IncludeCountry` and the `IncludeCounty` properties of this class can hide the `Country` and `County` fields. This method has the following signature:

```
format(address : AddressFillable, delimiter : String) : String
```

- The following version of the `format` method formats an address as text and includes only the specified set of fields. This method has the following signature:

```
format(address : AddressFillable, delimiter : String,
       fields : Set<AddressOwnerFieldId>) : String
```

The method parameters have the following meanings:

Parameter	Description
<code>address</code>	The address to format.
<code>delimiter</code>	Use this delimiter to separate the various string elements. If the delimiter is a comma, then the method also adds a space after the comma.
<code>fields</code>	The set of fields to include in the address.

PolicyCenter calls class `AddressFormatter` from the following places:

- From `entity.Address.DisplayName`. Use the **Entity Names** editor to modify the address string definition.
- From the `Address.addressString` method, defined in enhancement `AddressEnhancement`
- From the `PolicyLocation.addressString` method, defined in enhancement `PolicyLocationEnhancement`
- From the `PolicyAddress.addressString` method, defined in enhancement `PolicyAddressEnhancement`

Each of these uses creates a new `AddressFormatter` instance, populates the variables and then returns `AddressFormatter.addressString`.

In particular:

- `Address.DisplayName` always passes the same parameters. The other three uses just pass through the three parameters that they receive. This method always return a comma-delimited address string. However, this method is simpler to use as it requires no additional parameters.
- `Address.addressString` is the more general form of usage with an address. This is especially true if the `delimiter` value is something other than a comma.
- `PolicyLocation.addressString` and `PolicyAddress.addressString` work in the same manner as the `Address.addressString` method, except that they pass in the internally enhanced values for the address values. Other code in these enhancements determines whether to use the associated address values, or the internal values.

### See also

- “Additional Country and Address Configurations” on page 109

## Addresses and States or Jurisdictions

The country of an address controls the label used for the state or province field of an address through the `stateDisplayKey` setting in `country.xml` for that country. The `Jurisdiction` and `State` typelists have definitions for states, provinces, and other jurisdictions, each of which can be filtered.

Some examples:

- For Japan, PolicyCenter displays Kanji address fields.
- For Canada, PolicyCenter displays the label **Province** for this field.

### See also

- “Configuring the Country XML File” on page 108
- “Configuring Jurisdiction Information” on page 97

## Address Configuration Files

The following configuration files play a role in address configuration.

Studio location	File	Description
configuration → config → Extensions → Typelist	State.ttx	Used for addresses and locations.
	StateAbbreviation.ttx	Abbreviations used for states, provinces, and jurisdictions.
	Jurisdiction.ttx	Jurisdictions that regulate insurance and licensing. Similar to the State typelist.
	Country.ttx	Definitions of country codes for countries and regions.
configuration → config → geodata → <i>CountryCode</i>	address-config.xml – Defines formats to use for address auto-fill and input masks for postal codes.	Each country code has its own settings for these three files
	country.xml – See “Configuring the Country XML File” on page 108	
	zone-config.xml – See “Configuring Zone Information” on page 97	
configuration → config → geodata	<i>CountryCode</i> -locations.txt	Mappings between postal codes and cities 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 PolicyCenter uses to display addresses for specific countries. A country maps to an address mode by using the settings in `country.xml`.

If you add a new address format for a country or you add a new Address property, you must configure the files that support read-only addresses. See “Additional Country and Address Configurations” on page 109.

This topic includes:

- “Configuring the Country XML File” on page 108
- “Additional Country and Address Configurations” on page 109

## Configuring the Country XML File

PolicyCenter stores `country.xml` files in country-specific folders under the `geodata` folder, which you can access in Guidewire Studio. For example, the `country.xml` file for Japan is stored in the following folder:

`configuration → config → geodata → JP`

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

Attribute	Description	Related topic
<code>visibleFields</code>	Which address fields to display	"Visible Fields Attribute of the Country XML File" on page 108
<code>PCFMode</code>	Order in which to display address fields	"PCFMode Attribute of the Country XML File" on page 108
<code>postalCodeDisplayKey</code>	Label for the postal code field	"Postal Code Display Key Attribute of the Country XML File" on page 109
<code>stateDisplayKey</code>	Label for state or province field	"State Display Key Attribute of the Country XML File" on page 109

### Visible Fields Attribute of the Country XML File

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

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

The order of the fields does not matter. The fields must be defined in the class `AddressOwnerFieldId`.

See "AddressOwnerFieldId Class" on page 111.

If a `country.xml` file does not have `visibleFields` defined, PolicyCenter uses the set of address fields defined for the United States.

### PCFMode Attribute of the Country XML File

The attribute `PCFMode` of the `country.xml` file determines which modal version of the address PCF file that PolicyCenter uses for specific countries. For example, country-specific `country.xml` files individually specify the PCF mode for the following countries:

France	<code>PCFMode="PostCodeBeforeCity"</code>
Japan	<code>PCFMode="BigToSmall"</code>

If a `country.xml` file does not define the `PCFMode` attribute, PolicyCenter uses the default modal version of the address PCF file. In the base configuration, the default version is generally suitable for English-speaking countries.

#### See also

- "Address Modes in Page Configuration" on page 110

## Postal Code Display Key Attribute of the Country XML File

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

Japan	<code>postalCodeDisplayKey="Web.AddressBook.AddressInputSet.Postcode"</code>
United States	<code>postalCodeDisplayKey="Web.AddressBook.AddressInputSet.ZIP"</code>

If a `country.xml` file does not define `postalCodeDisplayKey`, PolicyCenter uses the value "Postcode".

## State Display Key Attribute of the Country XML File

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

Japan	<code>stateDisplayKey="Web.AddressBook.AddressInputSet.Prefecture"</code>
United States	<code>stateDisplayKey="Web.AddressBook.AddressInputSet.State"</code>

If a `country.xml` file does not have `stateDisplayKey` defined, PolicyCenter uses the value of state display key for the United States.

## Additional Country and Address Configurations

If you add a new country, in addition to the configurations described previously for the `country.xml` file, you must also add the country to a method of the `AddressFormatter` class. Add a new case option to the switch statement of the `AddressFormatter.internalFormat` method to handle the formatting of the address string for the new country.

If you extend the `Address` entity to add a new column, you must also incorporate this column into the following:

Class, enhancement, or configuration file	Task
<code>gw.address.AddressFormatter</code>	Add a new case option to the switch statement of the <code>AddressFormatter.internalFormat</code> method to handle the formatting of the address string for the new country.
<code>Address.en</code>	Modify the definition of the display name through the <b>Entity Names</b> editor. See "Using the Entity Names Editor" on page 125 in the <i>Configuration Guide</i> for details.
<code>AddressOwnerFieldID.gs</code>	Add a variable for the new <code>Address</code> column to this class. See "AddressOwnerFieldId Class" on page 111.
<code>gw.policylocation.PolicyLocationEnhancement</code>	<ol style="list-style-type: none"> <li>1. Modify the <code>PolicyLocation.addressString</code> method as needed and populate any added column before calling the <code>AddressFormatter</code> class.</li> <li>2. Add the get and set values for the new internal address columns.</li> </ol>
<code>gw.policyaddress.PolicyAddressEnhancement</code>	<ol style="list-style-type: none"> <li>1. Modify the <code>PolicyAddress.addressString</code> method as needed and populate any added column before calling the <code>AddressFormatter</code> class.</li> <li>2. Add the get and set values for the new internal address columns.</li> <li>3. Add the new columns to the <code>copyToNewAddress</code> and <code>copyFromAddress</code> functions.</li> </ol>

**See also**

- “AddressFormatter Class” on page 105

## Address Modes in Page Configuration

PolicyCenter uses a modal PCF file for all addresses, `GlobalAddressInputSet`. Modal versions of the global address PCF file determine the order of fields in addresses of specific countries.

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

Modal Address PCF File	Country
<code>GlobalAddressInputSet.BigToSmall</code>	<ul style="list-style-type: none"><li>• Japan</li></ul>
<code>GlobalAddressInputSet.PostCodeBeforeCity</code>	<ul style="list-style-type: none"><li>• France</li><li>• Germany</li></ul>
<code>GlobalAddressInputSet.default</code>	<ul style="list-style-type: none"><li>• Australia</li><li>• Canada</li><li>• Great Britain</li><li>• United States</li></ul>

To see the `GlobalAddressInputSet` PCF files, navigate in the Guidewire Studio **Project** window to **configuration** → **config** → **Page Configuration** → **pcf** → **address**.

### Mapping Countries to Modes

A country maps to a mode through the settings in `country.xml`.

### Controlling Field Properties

Each modal PCF file uses an implementation of the Gosu interface `AddressOwner` to control the following field properties:

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

### Gosu Address Formatter

PolicyCenter uses Gosu class `AddressFormatter` to format the address display fields. You can extend `AddressFormatter` to handle additional countries.

## Address Owners

`AddressOwner` is the interface for a helper object that is passed to the `GlobalAddressInputSet` PCF file. The helper object provides a way to set and get a single address on the enclosing entity. It also provides methods that you can use to set a field as required or visible, for example. Following are some properties on `AddressOwner`.

Property	Description
<code>Address</code>	Sets (or retrieves) a single address on the enclosing entity. For example, you can use this to set or get the primary address for a Contact. ClaimCenter automatically creates a new <code>Address</code> object if you use a Gosu expression of the form: <code>owner.Address.State = someState</code>
<code>HiddenFields</code>	Set of address fields that ClaimCenter hides (does not show) in the application interface.
<code>RequiredFields</code>	Set of address fields for which the user must supply a value.

In the base configuration, PolicyCenter provides the following classes that implement `AddressOwner` or extend a class that implements `AddressOwner`:

```
AddressOwnerBase
OptionalSelectedCountryAddressOwner
AccountAddressSearchOwner
AddressCountryCityStatePostalCodeOwner
ContactResultAddressSearchOwner
PolicyInfoAddressOwner
```

## AddressOwnerFieldId Class

Guidewire provides a `gw.api.address.AddressOwnerFieldId` class that provides type safety for `Address` entity fields. If you extend the `Address` entity with a new column, you must add the new column to this class as a new constant.

In the base configuration, `AddressOwnerFieldId` provides the following constants that represent address fields:

```
ADDRESSLINE1
ADDRESSLINE2
ADDRESSLINE1KANJI
ADDRESSLINE2KANJI
ADDRESSLINE3
ADDRESSTYPE
CEDEX
CEDEXBUREAU
CITY
CITYKANJI
COUNTRY
COUNTY
DESCRIPTION
POSTALCODE
STATE
VALIDUNTIL
```

The class `AddressOwnerFieldId` also defines a set of constants that use these address field ID constants. Some examples:

```
public final static var ALL_PCF_FIELDS : Set<AddressOwnerFieldId> =
{ ADDRESSLINE1, ADDRESSLINE2, ADDRESSLINE3, CITY, COUNTY, STATE, POSTALCODE, COUNTRY,
  ADDRESSLINE1KANJI, ADDRESSLINE2KANJI, CITYKANJI, CEDEX, CEDEXBUREAU }.freeze()

public final static var CITY_STATE_ZIP : Set<AddressOwnerFieldId> =
{ CITY, STATE, POSTALCODE, CEDEX, CEDEXBUREAU }.freeze()

public final static var HIDDEN_FOR_SEARCH : Set<AddressOwnerFieldId> =
{ ADDRESSLINE1, ADDRESSLINE2, ADDRESSLINE3, COUNTY, ADDRESSLINE1KANJI,
  ADDRESSLINE2KANJI, CEDEX, CEDEXBUREAU }.freeze()
```

## Automatic Address Completion and Fill-in

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

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

### Configuring Automatic Address Completion

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

For example, suppose that a user enters a postal code of 99501 for a United States address. If configured, PolicyCenter 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 by using 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, PolicyCenter 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 that 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, PolicyCenter 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 PolicyCenter. The base configuration of PolicyCenter provides the Java implementation class `DefaultAddressAutocompletePlugin`.

#### See also

- “Automatic Address Completion and Fill-in Plugin” on page 307 in the *Integration Guide*



# Configuring Phone Information

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

This topic includes:

- “Working with Phone Configuration Files” on page 113
- “Setting Phone Configuration Parameters” on page 114
- “Phone Number PCF Widget” on page 114

## Working with Phone Configuration Files

You can access phone-related configuration files in Studio by navigating in the **Project** window to **configuration** → **config** → **phone**.

Important files in this folder include the following:

File	Description
nanpa.properties	Area codes as defined by the North American Numbering Plan Administration (NANPA). These area codes apply to North American countries other than the United States.
PhoneNumberMetaData.xml	Area codes and validation rules for international phone numbers. Do not made changes to this file. See the comments at the beginning of the file for more information.
PhoneNumberAlternateFormats.xml	Additional area codes and validation rules for international phone numbers. Do not made changes to this file. See the comments at the beginning of the file for more information.

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

```
gwpc regen-phone-metadata
```

For details on how to use the gwpc commands, see “Build Tools” on page 110 in the *Installation Guide*.

## Setting Phone Configuration Parameters

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

Configuration parameter	Sets...
DefaultPhoneCountryCode	<p>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:</p> <p><a href="http://www.iso.org/iso/english_country_names_and_code_elements">http://www.iso.org/iso/english_country_names_and_code_elements</a></p> <p>The base application default phone country code is US.</p>
DefaultNANPACountryCode	<p>The default country code for region 1 phone numbers. If mapping file <code>nanpa.properties</code> does not contain the area code for this region, then PolicyCenter defaults to the area code value configured by this parameter.</p> <p>The base application default NANPA phone country code is US.</p>
AlwaysShowPhoneWidgetRegionCode	<p>Whether the phone number widget in PolicyCenter always shows a selector for phone region codes.</p> <p>The base application value for this parameter is <code>false</code>.</p>

See “Globalization Parameters” on page 56 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 PolicyCenter. The phone-related fields that you see in PolicyCenter depend on

- Country information that the user selects.
- Screen mode, editable or read-only.

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

- Country code
- National subscriber number
- Extension

If the screen is in read-only mode, a PolicyCenter user views previously entered phone-related information, the format of which depends 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 PolicyCenter from which the user can choose. For PolicyCenter to show the **Region Code** drop-down, the following must be true:

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

If the user initially enters a plus sign (+) in the phone number field, 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 by using `CountryCodeToRegionCode.xml` and identifies the corresponding country. PolicyCenter formats the phone number based on the user's phone region and the country selected for the phone number. For example:

User phone region	Phone number country	PolicyCenter 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 numeric phone number without first entering a country code, then PolicyCenter invokes only the format action on a targeted post-on-change event. Also, PolicyCenter invokes only the format action if the country code is the same as the user's selected default, or, if none, the default configured for the server.


#### See also

- “`AlwaysShowPhoneWidgetRegionCode`” on page 59 in the *Configuration Guide*

## Phone Numbers in 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 region selected by the user
- The default phone country code configured for the server

Users select a default phone region on the standard **Preferences** screen in PolicyCenter, available by clicking **Preferences** on the Options  menu. A user's selection for default phone region overrides the default phone region set for the server.



# Linguistic Search and Sort

This topic describes how to configure PolicyCenter to perform search and sort operations in languages other than the default en\_US. PolicyCenter provides support for language-appropriate text search and sort for a single language.

**Note:** This topic does not cover localization of free-text search. The base configuration of PolicyCenter provides free text search only in United States English. You must configure Guidewire Solr Extension to be able to perform free text searches in languages other than United States English. This configuration requires expertise in configuring Apache Solr. For information on basic configuration of free-text search, see “Free-text Search Configuration” on page 343 in the *Configuration Guide*.

This topic includes:

- “Linguistic Search and Sort of Character Data” on page 117
- “Effect of Character Data Storage Type on Searching and Sorting” on page 118
- “Searching and Sorting in Configured Languages” on page 118
- “Configuring Search in the PolicyCenter Database” on page 119
- “Configuring Sort in the PolicyCenter Database” on page 122

## Linguistic Search and Sort of Character Data

There are two primary ways to search for and sort character data:

- Treat the character data as binary code points and compare and sort the data numerically.
- Treat the character data linguistically. This approach applies specific collation rules to order words in a list that reflect the commonly accepted practices and expectations for a particular language.

Linguistic search applies a specific collation to the character data. A *collation* is an overriding set of rules that applies to the ordering and comparison of the data. Collation *strength* refers to the elements of the collation process that the search and sort code applies to the data.

For example:

- Collation strength controls whether the search and sort code respects or ignores differences in case and accent on a character, such as the leading character on a word.
- In the Japanese language, collation strength also controls whether the search and sort code respects or ignores the differences between:
  - Katakana and Hiragana.
  - Full-width and half-width Katakana character differences.

PolicyCenter uses the value of configuration parameter `DefaultApplicationLocale` and the `language.xml` and `collations.xml` configuration files to implement localized search and sort functionality.

**Note:** PolicyCenter displays the default locale value when the current locale value is missing. However, PolicyCenter always sorts by the current locale value. As a result, sorting of a localized column of a list view can appear to be broken when the column has one or more untranslated values.

## Effect of Character Data Storage Type on Searching and Sorting

Guidewire PolicyCenter stores character data in the following ways:

- Database storage
- In-memory storage

PolicyCenter handles searching and sorting of character data differently for these storage types.

### Character Data in the Database

PolicyCenter writes most application data directly to the database. This action stores the data on a physical disk storage system. Each discrete piece of data is an entry in a table column, with each table being organized by rows. During a comparison and sort of data in the database, the database management system (DBMS) performs the operations and applies rules that control these operations.

### Character Data in Memory

PolicyCenter writes some application data to volatile memory devices, such as the local machine RAM memory. PolicyCenter typically uses this kind of memory storage for the display of certain kinds of data in the user interface. For example, PolicyCenter uses in-memory storage for drop-down lists and the results of list views that do not use database queries. During a comparison and sort of data in memory, programming code provided in the base configuration controls the operations.

## Searching and Sorting in Configured Languages

Your PolicyCenter installation provides support for searching and sorting for a single language. PolicyCenter reads the localization code from the `DefaultApplicationLocale` configuration parameter, which you set in `config.xml`. The default is United States English, `en_US`.

You set the value of `DefaultApplicationLocale` once, before you start the application server for the first time. PolicyCenter stores this value in the database and checks the value at server startup. If the application server value does not match a database value, then PolicyCenter throws an error and refuses to start.

---

**IMPORTANT** You must set the value of configuration parameter `DefaultApplicationLocale` before you start the application server for the first time. You cannot change this value after you start the application server without dropping the database.

---

Guidewire also provides support for language-appropriate searching and sorting for display keys for each supported localization code. You define and manage language characteristics in `language.xml` files. You define these files for each localization folder, such as `en_US`. You can access the localization folders in Studio by navigating in the **Project** window to **configuration** → **config** → **Localizations**.

Each `language.xml` file contains a `<GWLanguage>` element. This element supports the following subelements that you can use to configure the behavior of searching and sorting operations in the Guidewire application:

- `<SortCollation>`  
Element `<SortCollation>` has a `strength` attribute that you use to define the sorting collation strength for this language. The exact meaning of the `strength` attribute value depends on the specific language. For more information on this attribute, see “Configuring Database Sort in `language.xml`” on page 123.
- `<LinguisticSearchCollation>`  
Element `<LinguisticSearchCollation>` supports a `strength` attribute that you use to define the searching collation strength for a language. For more information on this attribute, see “Configuring Oracle Search in `language.xml`” on page 119.

#### See also

- “Configuring Search in the PolicyCenter Database” on page 119
- “Configuring Sort in the PolicyCenter Database” on page 122
- “Setting Linguistic Search Collation” on page 57 in the *Upgrade Guide*

## Configuring Search in the PolicyCenter Database

For a column to be eligible for inclusion in the database search algorithm, the `supportsLinguisticSearch` attribute on that column must be set to `true`. Setting this column attribute to `true` marks that column as searchable, regardless of which DBMS you use.

See “`<column>`” on page 180 in the *Configuration Guide* for more information on the `<column>` element and the attributes that you can set on it.

---

**IMPORTANT** You cannot use the `supportsLinguisticSearch` attribute with an encrypted column. If you attempt to do so, the application server refuses to start.

---

How Guidewire PolicyCenter handles searching of data depends on the database involved. See the following:

- Searching and the Oracle Database
- Searching and the SQL Server Database

### Searching and the Oracle Database

To implement linguistic searching in Oracle, the database compares binary values that PolicyCenter modifies for searching. For functional and performance reasons, PolicyCenter does not use Oracle collations.

- For primary, accent-insensitive searching, Guidewire uses the configurable Java class described in “Configuring Oracle Search in `collations.xml`” on page 120 to compute the comparison values. Guidewire also uses this Java class to define the search semantics for searching in the Japanese and German languages.
- For secondary, case-insensitive searching, Guidewire transforms the search values to lower case.

#### Configuring Oracle Search in `language.xml`

Guidewire provides the ability to configure language-appropriate linguistic search capabilities through the `<LinguisticSearchCollation>` element. This subelement of `<GWLanguage>` is defined in `language.xml` in the

appropriate localization folder. You use the `strength` attribute of this subelement to configure and control specialized search behavior.

---

**IMPORTANT** Any change to the `<LinguisticSearchCollation>` element in `language.xml` requires a database upgrade. If you make a change to this element, then you must restart the application server to force a database upgrade.

---

The meaning of the `strength` attribute depends on the specific language. In general, the settings mean the following:

- A strength of `primary` considers only character weights. This setting instructs the search algorithms to consider just the base, or primary letter, and to ignore other attributes such as case or accents. Thus, the collation rules consider the characters `é` and `E` to have the same weight. For more information on this attribute, see “Configuring Database Sort in `language.xml`” on page 123.
- A strength of `secondary`, the default, considers character weight and accent differences, but, not case differences. Thus, the collation rules consider the characters `e` and `é` to be different and thus the rules treat them differently. The collation rules do not, however, treat `e` and `E` differently.

To summarize, the `strength` attribute can take the following values, with the default being `secondary`.

Strength	Search description
primary	<ul style="list-style-type: none"> <li>• accent-insensitive</li> <li>• case-insensitive</li> </ul>
secondary	<ul style="list-style-type: none"> <li>• accent-sensitive</li> <li>• case-insensitive</li> </ul>

**Note:** Localized search supports only two levels for the `strength` value, in contrast to localized sorting, which supports three levels for the `strength` value.

The following `language.xml` file is an example of this file in the localization folder `ja`, with suggested settings.

```
<?xml version="1.0" encoding="UTF-8"?>
<Language xmlns="http://guidewire.com/language">
  <GWLlanguage code="ja" name="Japanese" typecode="ja">
    <LinguisticSearchCollation strength="primary"/>
    <SortCollation strength="primary"/>
  </GWLlanguage>
</Localization>
```

## Configuring Oracle Search in `collations.xml`

For the Oracle database, Guidewire provides specialized search rules through the use of a Java class that you can configure. PolicyCenter exposes this Java class as a CDATA element in the `<Database>` element for Oracle in file `collations.xml`. You access `collations.xml` in the Studio Project window by navigating to **configuration** → **config** → **Localizations**.

In this file, search for the following:

```
<Database type="ORACLE">
  ...
  <DBJavaClass> <![CDATA[...]]></DBJavaClass>
  ...
</Database>
```

Guidewire PolicyCenter uses this Java code as the source code for a Java class. In the base configuration, the provided Java class defines:

- General rules for primary-strength searching in the database
- Specialized rules for searching in the Japanese language
- Specialized rules for searching in the German language



As defined in the comments in `collations.xml`, it is possible to modify the embedded `GNNormalize` Java class directly to meet your business needs. It is useful to modify the `GNNormalize` class under the following circumstances:

- You are using an Oracle database and either Japanese or German language strings
- You are using an Oracle database and primary, accent-insensitive search collation

### General Search Rules

In the base configuration, PolicyCenter uses the following general rules as it performs a database search on a column that is configured to support linguistic searching:

- All searches are case insensitive, regardless of the value of the `strength` attribute on `<LinguisticSearchCollation>`. PolicyCenter regards the characters `e` and `E` as the same.
- All searches take punctuation into account. PolicyCenter regards `O'Reilly` and `OReilly` as different.
- All searches in which the `strength` attribute on `<LinguisticSearchCollation>` is set to `primary` ignore accent marks. PolicyCenter regards the characters `e` and `è` as the same in this type of search.
- All searches in which the `strength` attribute on `<LinguisticSearchCollation>` is set to `secondary` take into account any accent marks. PolicyCenter regards the characters `e` and `è` as different in this type of search.

PolicyCenter searches only database columns for which you set the `supportsLinguisticSearch` attribute to `true`.

### General Search Rules for the Japanese Language

In the base configuration, Guidewire provides specialized search algorithms specifically for the Japanese language. Guidewire sets these rules in `collations.xml`, as described at the beginning of this topic. This Java class provides the following behavior for searching in a Japanese-language database:

Search case	Rule
Half-width/Full-width	All searches in Japanese ignore the difference between half-width and full-width Japanese characters.
Small/Large characters	All searches in Japanese in which the <code>strength</code> attribute on <code>&lt;LinguisticSearchCollation&gt;</code> is set to <code>primary</code> , meaning accent-insensitive, ignore Japanese small/large letter differences in Katakana or Hiragana. Searches in which this attribute is set to <code>secondary</code> take small/large letter differences into account.
Katakana and Hiragana	All searches in Japanese ignore the difference between Katakana and Hiragana characters. This type of search is known as <i>kana-insensitive</i> searching.
Long dash (—)	All searches in Japanese ignore the long dash character.
Sound marks (` and °)	All searches in Japanese in which the <code>strength</code> attribute on <code>&lt;LinguisticSearchCollation&gt;</code> is set to <code>primary</code> ignore sound marks. Searches in which this attribute is set to <code>secondary</code> take sound marks into account.

If you modify the contents of `collations.xml` or the embedded Java class, PolicyCenter forces a database upgrade the next time the application server starts.

### General Search Rules for the German Language

In the base configuration on Oracle, Guidewire provides specialized search algorithms specifically for the German language. Guidewire sets these rules through the use of a configurable Java class that it exposes as a CDATA element in the `collations.xml`. This is the same Java class that the discussion on rules for the Japanese language covered.

This Java class provides the following behavior for searching in a German-language database:

Search case	Rule
Vowels with umlauts	All searches in German compare as equal a vowel with an umlaut or the same vowel without the umlaut but followed by the letter e. Thus, all searches in German explicitly treat the following as the same value: <ul style="list-style-type: none"> <li>• ä and ae</li> <li>• ö and oe</li> <li>• ü and ue</li> </ul>
German letter Eszett	All searches in German treat the Eszett character ß (also known as Sharp-S) the same as the characters ss.

## Searching and the SQL Server Database

In SQL Server, the collations provided by the Windows operating system are effective in providing language-appropriate searching. To work correctly, Guidewire requires that you create a SQL Server database with case-insensitive collation. Guidewire uses this collation for all character data sorting and searching by default, as well as to provide case-insensitive table and column names.

Through the linguistic search configuration, it is possible to specify a different collation for searching on columns that support linguistic searching:

- If simple, case-insensitive searching meets your requirements, then configure file `collations.xml` to select the same collation as the database collation.
- If you need different search semantics, then configure the SQL Server entry in `collations.xml` for a primary strength search collation, which will give you accent-insensitive searching.

The semantics of linguistic searching for SQL Server are those of the Windows collation selected from the `collations.xml` file. The collation is based on the default language and linguistic search collation strength from `language.xml`, in which secondary strength is the default. Microsoft controls the Windows collation rules, not Guidewire.

With reference to the discussion about Japanese and German search rules on Oracle, the Windows collations configured in the base configuration in `collations.xml` provide the following:

- Kana-insensitivity and width-insensitivity for Japanese collations
- Umlaut and Eszett handling in the German collations

If you are currently using SQL Server in those languages, your IT staff is mostly likely familiar with these issues.

## Configuring Sort in the PolicyCenter Database

PolicyCenter handles the ordering of data as consistently as possible between database sorting and in-memory sorting. PolicyCenter derives the collation to use for sorting from the following:

- The default localization code set in the configuration parameter `DefaultApplicationLocale` in `config.xml`.
- The collation strength setting. This value is set in the localization folder's `language.xml` file.

PolicyCenter uses these values along with the database type to look up the collation in `collations.xml`.

**Note:** PolicyCenter uses in-memory sorting in the application interface for various elements, such as drop-down lists and list views that do not result from queries. To perform in-memory sorting, PolicyCenter uses a language-specific `Collator` object that is modified with the collation strength setting for that language.

This topic includes:

- “Configuring Database Sort in `language.xml`” on page 123

- “Configuring Database Sort in `collations.xml`” on page 123

## Configuring Database Sort in `language.xml`

For optional use, the `<SortCollation>` subelement of the `<GWLlanguage>` element in `language.xml` controls specialized sorting behavior. Each localization folder has a separate `language.xml` file. To access the localization folders, navigate in the Guidewire Studio **Projects** window to **configuration** → **config** → **Localizations** → *localizationFolder*.

The `<SortCollation>` element has a single `strength` attribute that determines *collation strength*—how PolicyCenter sorting algorithms handle accents and case during the sorting of character data for the following:

- Sorting of in-memory data
- Sorting of data in the database

The `strength` attribute, which defaults to `secondary`, can take the following values:

- `primary`
- `secondary`
- `tertiary`

The specific meaning of the `strength` attribute depends on the language. In general:

- A strength of `primary` instructs the search and sort algorithms to consider just the base, or primary letter, and to ignore other attributes, such as case or accents. With this setting, the collation rules consider the characters `e` and `E` to have the same weight.
- A strength of `secondary` instructs the search and sort algorithms to consider character weight and accent differences. This value is the default setting. With this setting, the collation rules consider the characters `e` and `è` to be different and order them differently.
- A strength of `tertiary` instructs the search and sort algorithms to consider character weight, accent differences, and case. With this setting, the collation rules consider the characters `e` and `è` and `E` to be different and order them differently.

The following list describes these differences.

Strength	Case-sensitive	Accent-sensitive
primary	No	No
secondary	No	Yes
tertiary	Yes	Yes

## Configuring Database Sort in `collations.xml`

Guidewire uses `collations.xml` as a lookup file. To access `collations.xml`, navigate in the Guidewire Studio **Projects** window to **configuration** → **config** → **Localizations**. PolicyCenter uses the following definitions in this file to look up the sort collation name and apply it:

- The application localization code
- The `strength` attribute value from the `<SortCollation>` element in `language.xml`
- The database management system (DBMS) type

PolicyCenter primarily uses these values to look up the sort collation. For example, suppose that the following are all true:

- The database is Oracle.
- The user language is German.
- The `strength` value of `SortCollation` in `language.xml` is set to `secondary`.

PolicyCenter then looks at the following for instructions on how to set NLS\_SORT for Oracle sessions and sets it to GERMAN\_CI.

```
<Database type="ORACLE">
...
  <Collation locale="de" primary="GERMAN_AI" secondary="GERMAN_CI" tertiary="GERMAN"/>
...
```

## Determining the Order of Typekeys

PolicyCenter uses the language collation rules defined in `language.xml` as part of determining the ordering of typekeys from the database. To sort typekeys, PolicyCenter applies the following criteria:

1. If there is no `.sort` file defined for the typelist in the localization folder, PolicyCenter:
  - a. Uses the priority associated with each typekey in its typelist to order the typekeys by priority order.
  - b. For typekeys with the same priority, applies the language collation rules to the typekey display names.
2. If there is a `.sort` file defined for the typelist in the localization folder, PolicyCenter:
  - a. Uses the order of the typekeys specified in that file.
  - b. For typekeys from the typelist that are not defined in the `.sort` file, orders them by applying the language collation rules to the typekey display names.

**Note:** You typically need the `.sort` file only if you are supporting Japanese with other languages on the same server. Otherwise, the preferred technique is to specify the sort order by defining priority in the typelist and language collation in `language.xml`.

PolicyCenter applies the collation rules to the typekey columns in database query ORDER BY clauses that sort database query results. File `collations.xml` contains multiple language collations because PolicyCenter supports storing typekey values in multiple languages in one database, enabling PolicyCenter to sort the typekey names correctly for each language. This storage scheme enables users with different language settings to see different translations of a typekey.

### See also

- For information on `.sort` files, see “Setting Localized Sort Orders for Localized Typecodes” on page 44.
- For information on setting typecode priority, see “Entering Typecodes” on page 269 in the *Configuration Guide*.

# Configuring National Field Validation

Field validation in PolicyCenter 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 125
- “Localizing Field Validators for National Field Validation” on page 126

**See also**

- “Field Validation” on page 257 in the *Configuration Guide*

## Understanding National Field Validation

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

- Field validators apply only to 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.

**Note:** You cannot define an input mask for input of Japanese characters—katakana and hiragana.

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. PolicyCenter applies national field validation based on the value of the country code associated with specific entity instances.

You configure field validation by editing `fieldvalidators.xml` files in various locations in the `fieldvalidators` folder.

- You define global field validators once in the `fieldvalidators.xml` file located in the root of the `fieldvalidators` folder.
- You define national field validators in `fieldvalidators.xml` files located in country-specific packages in the `fieldvalidators` folder.

See “Localizing Field Validators for National Field Validation” on page 126.

**See also**

- “Data Types” on page 231 in the *Configuration Guide*

## Localizing Field Validators for National Field Validation

You define national field validators in `fieldvalidators.xml` files located in country-specific folders in the `fieldvalidators` folder. Country-specific folder names must match typecodes from the Country typelist.

**To define national field validators for a specific country**

1. In Guidewire Studio, navigate in the **Project** window 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.

**See also**

- “Understanding National Field Validation” on page 125