Guidewire PolicyCenter™

Globalization Guide

Release 10.1.2

GUIDEWIRE
Adapt and succeed™

Product Name: Guidewire PolicyCenter
Product Release: 10.1.2
Document Name: Globalization Guide
Document Revision: 14-June-2021

Contents

Guidewire Documentation

About PolicyCenter documentation

The following table lists the documents in PolicyCenter documentation:

Document	Purpose
InsuranceSuite Guide	If you are new to Guidewire InsuranceSuite applications, read the *InsuranceSuite Guide* for information on the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with Guidewire applications.
Application Guide	If you are new to PolicyCenter or want to understand a feature, read the *Application Guide*. This guide describes features from a business perspective and provides links to other books as needed. The intended readers are everyone who works with PolicyCenter.
Upgrade Guide	Describes the overall upgrade process, and describes how to upgrade your configuration and database. The intended readers are implementation engineers who must merge base application changes into existing application extensions and integrations.
Configuration Upgrade Tools Guide	Describes the tools and functionality provided by the Guidewire InsuranceSuite Configuration Upgrade Tools. The intended readers are implementation engineers who must merge base application changes into existing application extensions and integrations. Visit the Guidewire Community to access the *Configuration Upgrade Tools Guide*, which is available for download, separately from the main documentation set, with the Configuration Upgrade Tools.
Installation Guide	Describes how to install PolicyCenter. The intended readers are everyone who installs the application for development or for production.
System Administration Guide	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.
Configuration Guide	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files for PolicyCenter. The intended readers are all IT staff and configuration engineers.
PCF Format Reference	Describes PolicyCenter PCF widgets and attributes. The intended readers are configuration engineers. See the *Configuration Guide*
Data Dictionary	Describes the PolicyCenter data model, including configuration extensions. The dictionary can be generated at any time to reflect the current PolicyCenter configuration. The intended readers are configuration engineers.
Security Dictionary	Describes all security permissions, roles, and the relationships among them. The dictionary can be generated at any time to reflect the current PolicyCenter configuration. The intended readers are configuration engineers.
Globalization Guide	Describes how to configure PolicyCenter for a global environment. Covers globalization topics such as global regions, languages, date and number formats, names, currencies, addresses, and phone numbers. The intended readers are configuration engineers who localize PolicyCenter.
Rules Guide	Describes business rule methodology and the rule sets in Guidewire Studio for PolicyCenter. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
Guidewire Contact Management Guide	Describes how to configure Guidewire InsuranceSuite applications to integrate with ContactManager and how to manage client and vendor contacts in a single system of record. The intended readers are PolicyCenter implementation engineers and ContactManager administrators.
Best Practices Guide	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
Integration Guide	Describes the integration architecture, concepts, and procedures for integrating PolicyCenter with external systems and extending application behavior with custom programming code. The intended

Document	Purpose
	readers are system architects and the integration programmers who write web services code or plugin code in Gosu or Java.
REST API Client Guide	Describes how to use the InsuranceSuite REST API Client to make outbound HTTP calls to internal or third-party REST services.
Java API Reference	Javadoc-style reference of PolicyCenter Java plugin interfaces, entity fields, and other utility classes. The intended readers are system architects and integration programmers.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Gosu API Reference	Javadoc-style reference of PolicyCenter Gosu classes and properties. The reference can be generated at any time to reflect the current PolicyCenter configuration. The intended readers are configuration engineers, system architects, and integration programmers.
ISBTF and GUnit Testing Guide	Describes the tools and functionality provided by InsuranceSuite for testing application behavior during an initial implementation or an upgrade. The guide covers functionality related to Behavior Testing Framework, GUnit, and Gosu functionality designed specifically for application testing. There are two sets of intended readers: business analysts who will assist in writing tests that describe the desired application behavior; and technical developers who will write implementation code that executes the tests.
Glossary	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.
Advanced Product Designer Guide	Advanced Product Designer is a tool that helps you, a business user, design, simulate, and deploy an insurance product. The intended readers are business analysts who understand insurance products, business systems analysts who liaise between business analysts and IT, project managers, and IT who provides technical expertise in areas such as programming, testing, and databases.
Product Model Guide	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 *Product Designer Guide*.
Product Designer Guide	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.
REST API Framework	Describes the Guidewire InsuranceSuite framework that provides the means to define, implement, and publish REST API contracts. It also describes how the Guidewire REST framework interacts with JSON and Swagger objects. The intended readers are system architects and integration programmers who write web services code or plugin code in Gosu or Java.

Conventions in this document

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

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

narrow bold	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Click **Submit**.

Text style	Meaning	Examples
`monospace`	Code examples, computer output, class and method names, URLs, parameter names, string literals, and other objects that might appear in programming code.	The `getName` method of the `IDoStuff` API returns the name of the object.
`monospace italic`	Variable placeholder text within code examples, command examples, file paths, and URLs.	Run the `startServer` *`server_name`* command. Navigate to `http://`*`server_name`*`/index.html`.

Support

For assistance, visit the Guidewire Community.

Guidewire customers

`https://community.guidewire.com`

Guidewire partners

`https://partner.guidewire.com`

chapter 1

Understanding globalization

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

Dimensions of globalization

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

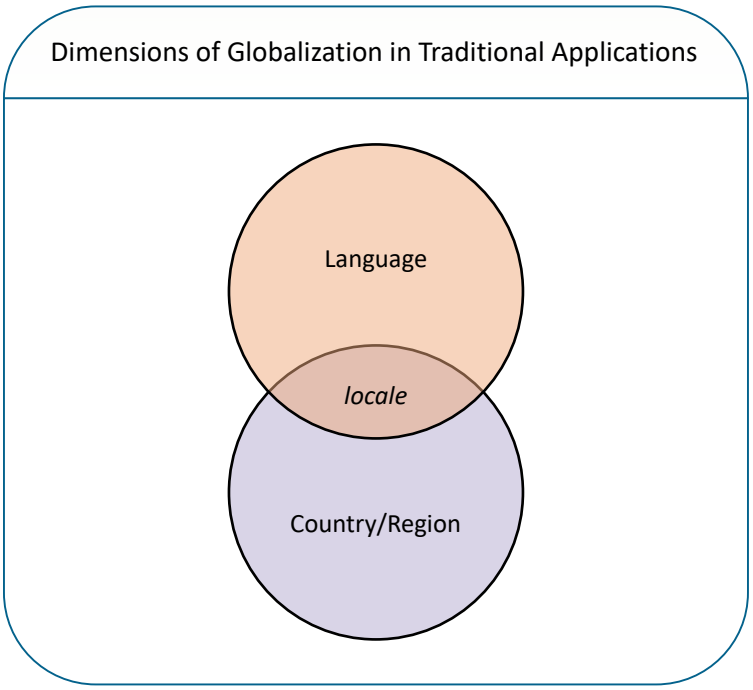
Language

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 preconfigured 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, language and country/region and their intersection as locale 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 predefined 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.

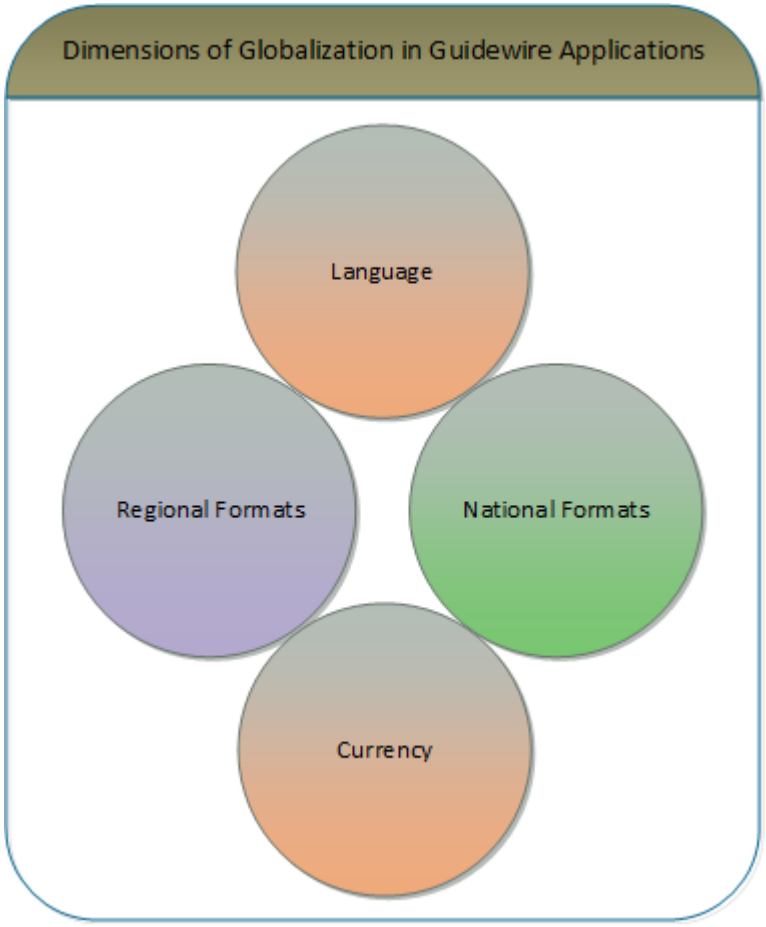
National formats

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.

Currency formats

Currency is independent of language, locale, and national formats. A specific currency is a property of a specific PolicyCenter transaction. The values set for locale and regional formats determine how to format the currency amount. However, visual formatting has no impact on the actual currency used in the transaction.

For example, a speaker of Canadian French can be operating in an `en-CA` locale looking to insure property in Florida in USD rather Canadian dollars.



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.

However, phone numbers and addresses in PolicyCenter use national (country) formatting, set through application configuration. For example, if you enter the Japan country code +81 in a phone field, PolicyCenter displays the phone number by using Japanese 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 or language.

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

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

Options for setting the display language

In the base configuration, Guidewire configures PolicyCenter to use a single display language, which is United States English. To view another language, you must enable at least one additional language and configure PolicyCenter for that language.

If your installation has more than one configured language, you can select among the available choices using the PolicyCenter **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 and your user administrator has not set your language, PolicyCenter uses the language specified by your web browser. Configuration parameter `DefaultApplicationLanguage` specifies the primary language for PolicyCenter screens, but it does not specify the default browser language. In the base configuration, Guidewire sets the primary language to U.S. English.

Options for setting 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 on the **Regional Formats** menu. The base configuration defines the following locale types:

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

The default regional format for a user is set in the profile of that user on the **Administration** tab.

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.

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.

configuration→config

`config.xml`

Configuration parameters related to localization. The localization-related configuration parameters include, among others:

- `DefaultApplicationLocale`
- `DefaultApplicationLanguage`
- `DefaultCountryCode`
- `DefaultPhoneCountryCode`

Each Guidewire application instance contains a single copy of `config.xml`.

configuration→config→Extensions→Typelist

`Country.ttx`, `Country.default.ttx`

Country codes that reflect the Unicode Common Locale Data Repository (CLDR) country names.

`Currency.example.ttx`

Contains examples of currency code and similar information for the supported currencies.

`Jurisdiction.ttx`, `Jurisdiction.example.ttx`

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.

`LanguageType.ttx`

List of languages that you want to have enabled in PolicyCenter.

IMPORTANT During development, if you make a change that requires that a language typecode in this typelist not be in effect when you restart, you must retire the language typecode. Do not delete the language typecode, or you might have database errors when you restart the application.

`PhoneCountryCode.ttx`

Phone country codes.

`State.ttx`

Address configuration.

`StateAbbreviation.ttx`, `StateAbbreviation.example.ttx`

Defines abbreviations of the names of states,provinces, territories, and so on by country.

configuration→config→Localizations

`collations.xml`

Collation configuration for one or more languages that apply to one or more database types.

`localization_`*`localeCode`*`.xml`

(Example: `localization_en_US.xml`) Currency formatting information for use with single currency rendering mode only. Defines other region settings like date and time formats and numeric separators. Studio supports multiple localization files.

`language_languageCode.xml`

(Example: `language_de_DE.xml`) Contains collation definitions, input field label width, and date-time field hints for a specific language.

configuration→config→Localizations→Resource Bundle 'display'

`display_languageCode.properties`

Application display keys for a specific language. Each region must have a separate `display_languageCode.properties` file. A `display_languageCode.properties` file defines standard Java properties by using the following format:

```
display_key_name = value
```

`display.properties`

Application display keys for the fallback language. In the base configuration, this file contains display keys for U.S. English.

configuration→config→Localizations→Resource Bundle 'typelist'

`typelist_languageCode.properties`

(Example: `typelist_de.properties`) Application typecode names and descriptions for a specific language.

`typelist.properties`

Fallback language for typelist properties. In the base configuration, this file contains U.S. English typelist properties.

configuration→config→Localizations→Resource Bundle 'gosu.display'

`gosu.display_languageCode.properties`

(Example: `gosu.display_de.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.

`gosu.display.properties`

Fallback language for Gosu properties. In the base configuration, this file contains U.S. English values for Gosu properties.

configuration→config→Localizations→Resource Bundle 'productmodel.display'

`productmodel.display_languageCode.properties`

(Example: `productmodel.display_de.properties`) Display keys for the fields used in the PolicyCenter product model. Except for U.S. English, the text strings in this file are in the language specified by *localeCode*.

`productmodel.display.properties`

Fallback language for product model display properties. In the base configuration, this file contains U.S. English values for product model display keys.

configuration→config→Metadata→Typelist

`LocaleType.tti`

List of defined regions.`LocaleType.tti`

`PhoneCountryCode.tti`

List of country code values for phones.

configuration→config→currencies *currencyCodeFolder*

`currency.xml`

Regional format overrides for monetary amounts in installations with multiple currencies.

configuration→config→datatypes

`*.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. Does not apply to date-time fields.

configuration→config→fieldvalidators *countryCodeFolder*

`fieldvalidators.xml`

Field validation definition overrides by country.

configuration→config→geodata

countryCode`-locations.txt`

Mapping between postal codes and cities for a country.

configuration→config→geodata *countryCodeFolder*

`address-config.xml`

Address configuration by country.

`country.xml`

Visible address fields and display keys for the state and postal code labels.

`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. See the comments at the beginning of the file for more information.

`PhoneNumberAlternateFormats.xml`

Additional area codes and validation rules for international phone numbers. See the comments at the beginning of the file for more information.

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 the *Configuration Guide*.

Parameter name	Description
`AlwaysShowPhoneWidgetRegionCode`	Whether the phone number widget in PolicyCenter displays a selector for phone region codes.
`DefaultApplicationCurrency`	Currency to use by default for new monetary amounts or whenever the use does not specify a currency for an amount.
`DefaultApplicationLanguage`	Primary language for field labels and other string resources that PolicyCenter shows to users who do not select a personal language preference. The language must be enabled, and the specified language displays only for display keys that are defined for that language. Undefined display keys in the primary language cause the application to use language fallback.
`DefaultApplicationLocale`	Region for regional formats in the application, unless the user selects a different personal preference for regional formats.

Parameter name	Description
DefaultCountryCode	Country code to use for new addresses by default or if a user does not specify a country code for an address explicitly.
DefaultNANPACountryCode	Default country code for region 1 phone numbers. NANPA stands for North American Numbering Plan Administration.
DefaultPhoneCountryCode	Server-wide country code to use for new phone numbers by default or if a user does not specify the country code during phone number entry.
DefaultRoundingMode	Default rounding mode for monetary amount calculations. Guidewire recommends limiting this setting to HALF_UP or HALF_EVEN.
MulticurrencyDisplayMode	When set to MULTIPLE, PolicyCenter displays a currency selector for monetary amounts. In single currency installations, setting SINGLE, there is no need for a currency selector because all amounts are in the default currency.

chapter 2

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. Typically, you configure PolicyCenter for display languages by enabling languages supplied by Guidewire.

Working with display languages

In the base configuration, the primary display language in PolicyCenter is United States English, for which string resources are defined in `display.properties` and `typelist.properties`.

To display languages other than U.S. English, you must do the following:

Enable additional display languages

When you configure PolicyCenter to use a language provided by Guidewire, PolicyCenter configures localized screen and field labels and other string resources for that language.

Select the primary display language for the application

Set one of the enabled languages as the primary display language for PolicyCenter in the `DefaultApplicationLanguage` configuration parameter. The *primary display language* is the default preferred language for users. New users are assigned the primary display language as their default preferred language. They see user interface labels, window names, and so on in the primary language at the time that they log in. A user can select any language that has been enabled in PolicyCenter as their preferred display language.

Note: Guidewire PolicyCenter uses language fallback to display string resources, such as display keys and typecode keys, that are not defined for the user's preferred display language.

Important:

PolicyCenter enables you to configure display languages of your choice manually. However, Guidewire does not provide language configuration support for configuring PolicyCenter with a language that Guidewire does not provide.

If you manually configure your own language and have questions, contact your Guidewire Professional Services representative.

See also

- “About language fallback” on page 20
- “Enabling display languages” on page 21
- “Setting the primary display language” on page 23

About language fallback

Guidewire PolicyCenter uses a *language fallback* hierarchy for display languages. The fallback mechanism:

- Orders the set of language property files into a strict hierarchy.
- Uses language and locale codes in the property file names to indicate the fallback hierarchy.

In general, you enable a language and set it as the primary display language, and, if necessary, add a display key property file and a typelist property file for that language. When you run the application, language fallback occurs for display keys and typecode keys that are missing.

Thus, language fallback means that if a display key or typelist key is not available in the user's preferred language, PolicyCenter uses the keys defined in the next property file in the hierarchy, if there is one. For example, `display_fr_CA.properties` falls back to `display_fr.properties`, which, in turn, falls back to the base language defined in `display.properties`.

Language fallback always applies to the user's preferred language. For example, if, in addition to Canadian French, you enable Spanish for PolicyCenter, a user can change the preferred language from Canadian French to Spanish. For that user, if a display key is missing for Spanish, the language falls back to the language defined in `display.properties`, which, in the default configuration, is U.S. English.

Note: Setting the primary display language in configuration parameter `DefaultApplicationLanguage` does not cause that language to become a fallback for any other language. Setting the primary display language affects only the default preferred language for users. A user always has a preferred language. The default for that preferred language is the language defined as the primary display language.

Example fallback for French localization

In the base configuration, Guidewire provides standard French display key and typelist property files:

- `display_fr.properties`
- `typelist_fr.properties`

However, suppose that you want Canadian French to be the language that users see when they log into PolicyCenter. To accomplish this task, you need to perform the following work flow.

1. Define the French Canadian display keys that differ from standard French in file `display_fr_CA.properties`, located in the following Studio directory:

   configuration→config→Localizations→Resource Bundle 'display'
2. Define the French Canadian typecode keys that differ from standard French in file `typelist_fr_CA.properties`, located in the following Studio directory:

   configuration→config→Localizations→Resource Bundle 'typelist'
3. Enable a French Canadian language option by adding the following entry to file `LanguageType.ttx`, located in the following Studio directory:

   configuration→config→Extensions→typelist

   ```
   <typecode code="fr_CA" desc="Français Canadien" name="Français Canadien"/>
   ```
4. Set configuration parameter `DefaultApplicationLanguage` to `fr_CA` in file `config.xml`.
5. Restart the application server to enable these changes in Guidewire PolicyCenter.

Thereafter, whenever a user logs into PolicyCenter, the user sees Canadian French as the default language, unless the user, at some point, changes the preferred display language to a different enabled language in PolicyCenter.

For a user who is using Canadian French, if PolicyCenter cannot find a Canadian French display key for an element in the user interface, the display language falls back first to the definition for French in `display_fr.properties`. If this file cannot provide a value for the display key, the display language falls back to the definition in `display.properties`. The same happens for typecode keys.

Synchronizing language property files

In the base configuration of PolicyCenter, Guidewire provides U.S. English as the default language defined in files `display.properties` and `typelist.properties`. However, if you do not intend to use U.S. English as an

available language in PolicyCenter, it is not necessary to maintain these files in synchronization with additional language display keys and typecodes that you add to other language files. You are free to do so, but Guidewire does not require it.

Enabling display languages

To use a display language other than U.S. English, open Guidewire Studio™ for PolicyCenter and configure the `LanguageType` typelist to use a supported language. Then start the server.

On server startup, the PolicyCenter configures localized screen and field labels and other string resources for that language. After the server starts, the language is available.

See also

- "Setting the primary display language" on page 23
- "Selecting a personal language preference" on page 23
- "Upgrading display languages" on page 24

General language enablement considerations

If you enable other languages in addition to the base configuration default (U.S. English), it is possible to make one of these additional languages the primary language for your Guidewire application. To set the primary language, set configuration parameter `DefaultApplicationLanguage` in file `config.xml`.

How PolicyCenter decides which language to present to the user

1. Initially, you configure PolicyCenter to support one or more languages, as defined in the `LanguageType` typelist.
2. You then set one of the defined languages as the default application language by setting configuration parameter `DefaultApplicationLanguage` to your preferred value.
3. At runtime, the browser in which the application runs sends a list of one or more preferred languages in the HTTP request's Accept-Language header. It is possible to configure the browser with the set of preferred languages.
4. PolicyCenter then attempts to show the user interface in the user's preferred language, as defined in the Accept-Language header.
5. If PolicyCenter does not support any of the languages passed in the Accept-Language header, PolicyCenter then attempts to show the user interface in the language defined as the default application language (`DefaultApplicationLanguage`).

The `DefaultApplicationLanguage` language is also the one you configure for data that does not use display keys or typecode keys. You manually configure this data in only one language, the primary language.

Setting the primary language and default locale

If you enable a display language, you might also want to set the `DefaultApplicationLanguage` and `DefaultApplicationLocale` configuration parameters in `config.xml` to that of your target language and region.

Note: If you do not make this change, certain items in the application user interface will remain in the primary language and default locale, which in the base configuration is U.S. English.

You can set these two configuration parameters only to a supported language and region, as follows:

- You can set `DefaultApplicationLanguage` to an enabled language.
- You can set `DefaultApplicationLocale` either to a locale that PolicyCenter supports in the base configuration or to the locale of an installed region pack.

IMPORTANT To be able to set the value of `DefaultApplicationLanguage` or `DefaultApplicationLocale` and have it affect an application, you must set these values and then start the server for the first time.

See also

- "Setting the primary display language" on page 23

Setting the default currency

You can change the default currency from USD to your local currency by setting `DefaultApplicationCurrency` in file `config.xml`.

You must set `DefaultApplicationCurrency` and perform additional configuration on the product model.

IMPORTANT To apply a new value of `DefaultApplicationCurrency`, you must drop the database before you restart the server.

See also

- For additional information on configuring currencies, see the *Configuration Guide*

Enable a display language

About this task

All display languages that are provided by Guidewire are available in Guidewire Studio™. To enable a language, use Studio to indicate that a language is to be used in the product, and then restart the product.

Procedure

1. Stop the PolicyCenter application server.
2. Start Guidewire Studio for PolicyCenter.
3. In the Studio **Project** window, navigate to the following location in the Settings dialog.

   File→Settings→Guidewire Studio
4. Set the **Show filename suffix on new extension dialog** check box.
5. In the Studio **Project** window, navigate to the following location:

   configuration→config→Extensions→Typelist
6. Selecting **Typelist**, use the right-click contextual menu to create a new `LanguageType` typelist extension, entering the correct language/locale code for the language that you want to enable in GenericCenter in the **Filename Suffix** entry field at the bottom of the dialog, `fr_FR`, for example.
7. In the new typelist extension, add the corresponding typecode, along with the name and description of the language.
8. Start the application server.

   After the server starts, the language that you enabled is now a choice from the **Options** (gear icon) menu, for example:

   Options→International→Language→French (FR)

View supported language files in Studio

All translated languages provided by Guidewire are available in Guidewire Studio™.

Procedure

1. Start Guidewire Studio.
2. To see the display language files, navigate in the **Project** window to **configuration→config→Localizations→Resource Bundle 'display'**.

   The default U.S. English file is `display.properties`.
3. Navigate back up to **Localizations** to see additional resource bundles and the collations, language, and localization files. For example:
   - `collations.xml`
   - **Resource bundle 'display'**
   - **Resource bundle 'gosu.display'**
   - `language_de_DE.xml`
   - `language_ja.xml`
   - `language_ja_JP.xml`
   - `localization_en_US.xml`
   - `localization_fr_FR.xml`
   - `localization_ja_JP.xml`
   - **Resource bundle 'productmodel.display'**
   - **Resource bundle 'typelist'**

Setting the primary display language

The `DefaultApplicationLanguage` configuration parameter in `config.xml` sets the *primary display language*, the default preferred language for users. New users are assigned the primary display language as their default preferred language. They see user interface labels, window names, and so on in the primary language at the time that they log in. A user can select any language that has been enabled in PolicyCenter as their preferred display language.

You must enable the language that you want to use for `DefaultApplicationLanguage`. Set the value of this configuration parameter before you start your PolicyCenter server for the first time.

IMPORTANT You can enable additional display languages later, but you cannot change the primary application language after starting the server, even if you restart it.

The value that you set for `DefaultApplicationLanguage` must be a typecode in the `LanguageType` typelist. If you set the value of the parameter `DefaultApplicationLanguage` to a value that is not a `LanguageType` typecode, the application server cannot start.

See also

- "Enable a display language " on page 22
- "About language fallback" on page 20
- *Configuration Guide*

Selecting a personal language preference

Users of PolicyCenter can choose a preferred display language from the **Options** (gear icon) menu by navigating to **International→Languages**. Language choices are available only for enabled languages. A user's language preference overrides the primary application language that you set system-wide with the 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 primary display language” on page 23
- “Selecting language and regional formats in PolicyCenter” on page 14

Upgrading display languages

See the Guidewire *Upgrade Guide* for a description of the process involved in upgrading display languages between different PolicyCenter versions. The *Upgrade Guide* is available for download with the Guidewire InsuranceSuite Upgrade Tools.

chapter 3

Localized printing

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

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 Portable Document Format (PDF) 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, you might need to use specialized font families to handle languages such as Japanese.

Guidewire does not provide fonts for use with Guidewire products. Any fonts that you use must be provided by a vendor for your operating system platform. 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:

- You must ensure that appropriate fonts are available for document printing and other features of the Guidewire applications.
- Document fonts must be provided and supported by the operating system vendor.
- Guidewire does not guarantee any of the fonts supplied as part of an operating system platform.

Printing with non-standard fonts

It is possible to print PDF documents from PolicyCenter that use non-standard fonts. However, to do so, your operating system must support the required font. Keep in mind that it is also possible for different versions of an operating system to change support for various fonts. For example:

- Microsoft Windows 7 supports Japanese font `msmincho.ttc` and the font exists in the default Windows 7 configuration.
- Microsoft Windows 10 no longer supports the `msmincho.ttc` font and other non-English fonts in the basic configuration. However, Windows 10 users can add additional fonts as an optional feature.

Microsoft provides instructions on how to install missing fonts after an upgrade to Windows 10 in the following web site:

`https://docs.microsoft.com/en-us/windows/deployment/windows-10-missing-fonts`

Also keep in mind that the Japanese-language version of the Windows 10 operating system can contain various fonts by default that are not available in the standard English-language version of the Windows operating system.

Localized printing in a Windows environment

Printing PDF documents that require a character set not supported by PDF requires additional setup. For example, the Russian language uses the Cyrillic character set. The default fonts for PDF generation do not support the Cyrillic character set. Therefore, you must customize the Apache Formatting Objects Processor (FOP) application to use fonts that do support the Cyrillic character set.

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

The example assumes the following:

- Apache FOP is installed on your machine.
- The `fop.jar` file is on the class path.
- The Arial fonts are in `C:\WINDOWS\Fonts`.
- The fonts are TrueType fonts.
- You have a working Guidewire PolicyCenter application.
- You have enabled the proper language for your Guidewire application.

  Note: The process for configuring FOP supported fonts that are not TrueType is different. See the Apache FOP documentation for more information.

Register the fonts with Apache FOP

About this task

The default configuration of Adobe FOP (and the fonts that it uses to build output PDFs) supports only the Latin1 character set, used by languages such as English, German, French, Italian, Spanish, Portuguese, and Dutch. The default configuration does not support languages with non-Latin alphabets such as Russian (Cyrillic alphabet), Japanese, and Chinese. Thus, you must specifically configure the `fop.xconf` file to support the Russian, Japanese, and Chinese alphabets.

Procedure

1. Copy the following file into directory `C:/fopconfig/`:

   `C:/fop/conf/fop.xconf`

2. Open `fop.xconf` for editing and find the following code in the `<font>` section.

```
<!-- auto-detect fonts -->
<auto-detect/>
```

3. Remove the existing `<auto-detect/>` element and replace it with the following code.

```
<!-- Japanese -->
<font auto-detect="yes" kerning="yes" embed-url="file:///C:/windows/fonts/msmincho.ttc" sub-font="MS Mincho">
  <font-triplet name="Japanese" style="normal" weight="normal"/>
  <font-triplet name="Japanese" style="normal" weight="bold"/>
  <font-triplet name="Japanese" style="italic" weight="normal"/>
  <font-triplet name="Japanese" style="italic" weight="bold"/>
</font>

<!-- Chinese -->
<font auto-detect="yes" kerning="yes" embed-url="file:///C:/windows/fonts/simsun.ttc" sub-font="SimSun">
  <font-triplet name="Chinese" style="normal" weight="normal"/>
  <font-triplet name="Chinese" style="normal" weight="bold"/>
  <font-triplet name="Chinese" style="italic" weight="normal"/>
  <font-triplet name="Chinese" style="italic" weight="bold"/>
</font>
```

```
<!-- Russian -->
<font auto-detect="yes" kerning="yes" embed-url="file:///C:/windows/fonts/arial.ttf">
  <font-triplet name="Cyrillic" style="normal" weight="normal"/>
</font>

<font auto-detect="yes" kerning="yes" embed-url="file:///C:/windows/fonts/arialbd.ttf">
  <font-triplet name="Cyrillic" style="normal" weight="bold"/>
</font>

<font auto-detect="yes" kerning="yes" embed-url="file:///C:/windows/fonts/ariali.ttf">
  <font-triplet name="Cyrillic" style="italic" weight="normal"/>
</font>

<font auto-detect="yes" kerning="yes" embed-url="file:///C:/windows/fonts/arialbi.ttf">
  <font-triplet name="Cyrillic" style="italic" weight="bold"/>
</font>
```

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

Next steps

The next step is “Register FOP configuration and font family with PolicyCenter” on page 27.

Register FOP configuration and font family with PolicyCenter

Before you begin

Complete the step “Register the fonts with Apache FOP” on page 26 before you perform this step.

About this task

You must specifically add the non-Latin1 languages that you want to support in file `config.xml`.

Procedure

1. Open Guidewire Studio™ for PolicyCenter and press `Ctrl+Shift+N`, and then search for `config.xml`.
2. Open `config.xml`, find parameter `PrintFontFamilyName` and set it to the following value.

   ```
   <paramname="PrintFontFamilyName"value="sans-serif, Cyrillic, Japanese, Chinese"/>
   ```

   Do not change the `san-serif` designation as this would affect the Latin1 languages. List the non-Latin1 languages in the order shown, with Russian (Cyrillic) first.
3. Find parameter `PrintFOPUserConfigFile` and set it to the following value.

   ```
   <paramname="PrintFOPUserConfigFile"value="C:/fopconfig/fop.xconf"/>
   ```

   This configuration parameter provides a fully qualified path to a valid FOP user configuration file. The parameter must point to the `fop.xconf` file that you modified.
4. Stop and start the PolicyCenter server so that these changes can take effect.

Next steps

The next step is “Testing your configuration” on page 27.

Testing your configuration

Before you begin

Complete the step “Register FOP configuration and font family with PolicyCenter” on page 27 before you perform this step.

About this task

After you perform the steps to localize printing in a Microsoft Windows environment, 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 Russian.

Localized printing in a Linux environment

This example describes how to configure Guidewire PolicyCenter and the Apache Formatting Objects Processor (FOP) to print Japanese characters in a Linux environment.

The example assumes the following:

- You have a working Guidewire PolicyCenter application.
- You have enabled the proper language for your Guidewire application.

A package manager is recommended for managing 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

Choose a package manager that works with your particular Linux distribution.

Download and install the required fonts

Before you begin

You must obtain and install a font that supports the language in which you want to print. For example, to print Japanese characters, 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

Procedure

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

   For example, you can obtain the Sazanami Gothic font from one of the following Japanese-language web sites:

   https://osdn.net/projects/efont/releases/10087

   https://ja.osdn.net/projects/efont/
2. After downloading the font file, untar the file and extract `sazanami-gothic.ttf` to the file directory of your choice.

Next steps

The next step is “Configure the font” on page 28.

Configure the font

Before you begin

Complete the step “Download and install the required fonts” on page 28 before you perform this step.

About this task

This procedure assumes the use of Japanese font Sazanami Gothic.

Procedure

1. Copy the font that you are installing into the following directory on the Linux machine:

   `/usr/local/fop`
2. Navigate to directory `/user/local/fop`, if you are not there already.
3. Run the following command in this directory:

```
java -cp build/fop.jar:lib/avalon-framework-4.2.0.jar:lib/commons-logging-1.0.4.jar:
    lib/xmlgraphics-commons-2.3.jar:lib/commons-io-1.3.1.jar
    org.apache.fop.fonts.apps.TTFReader
    -ttcname "Sazanami Gothic" sazanami-gothic.ttf sazanami-gothic.xml
```

   The example introduces line spacing for readability. In actual practice, this is a single long command string.

   This action creates configuration file `sazanami-gothic.xml` in directory `usr/local/fop`.
4. Move, or copy, files `sazanami-gothic.ttf` and `sazanami-gothic.xml` into the `conf` subdirectory.

Next steps

The next step is “Register the font with Apache FOP” on page 29.

Register the font with Apache FOP

Before you begin

Complete the step “Configure the font” on page 28 before you perform this step.

About this task

The default configuration of Adobe FOP (and the fonts that it uses to build output PDFs) supports the Latin1 character set only. Languages such as English, German, French, Italian, Spanish, Portuguese, and Dutch use the Latin1 character set. The default configuration does not support languages with non-Latin alphabets such as Russian (Cyrillic alphabet), Japanese, and Chinese. Thus, you must specifically configure the `fop.xconf` file to support the Russian, Japanese, and Chinese alpahbets.

Procedure

1. Open `fop.xconf` for editing.

   To use the `vi` editor, enter the following at a command prompt:

```
vi conf/fop.xconf
```
2. Find the following code in the <fonts> section:

```
<!-- auto-detect fonts -->
<auto-detect>
```
3. Immediately above the `<auto-detect>` section, add something similar to the following example for each non-Latin1 language that you want to support.

```
<font metrics-url="sazanami-gothic.xml" kerning="yes" embed-url="sazanami-gothic.ttf">
  <font-triplet name="Japanese" style="normal" weight="normal">
  <font-triplet name="Japanese" style="normal" weight="bold">
  <font-triplet name="Japanese" style="italic" weight="normal">
  <font-triplet name="Japanese" style="italic" weight="bold">
<font>
```

   Adding the `embed-url` attribute embeds the named font in the generated PDF document. This enables one to view and properly display the PDF on an operating system that does not contain the necessary font.

Next steps

The next step is “Register FOP configuration and font family with PolicyCenter” on page 30.

Register FOP configuration and font family with PolicyCenter

Before you begin

Complete the step "Register the font with Apache FOP" on page 29 before you perform this step.

About this task

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

Procedure

1. Open Guidewire Studio™ for PolicyCenter and search for `config.xml`.
2. Find the parameters `PrintFontFamilyName` and `PrintFOPUserConfigFile` in `config.xml` and set them accordingly.

   The following example registers Japanese-language fonts with Guidewire PolicyCenter.

Parameter	Description	Example value
`PrintFontFamilyName`	The name of the language, or languages, that PolicyCenter is to support.	`<param name="PrintFontFamilyName" value="sans-serif, Japanese"/>`
`PrintFOPUserConfigFile`	The fully qualified path to a valid FOP configuration file.	`/usr/local/fop/fopconfig/`

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

Next steps

The next step is "Test your configuration" on page 30.

Test your configuration

Before you begin

Complete the step "Register FOP configuration and font family with PolicyCenter" on page 30 before you perform this step. To test the Apache FOP configuration, you must have a PolicyCenter implementation that supports the Japanese locale.

About this task

After you perform the steps to localize printing in a Linux environment, test that you are able to create and print a PDF file that uses the correct font.

Procedure

1. Open Guidewire PolicyCenter.
2. Navigate to a PolicyCenter screen that contains localized strings.
3. Generate and review a localized PDF to verify that the PDF displays the font correctly.

chapter 4

Localizing PolicyCenter string resources

This topic describes how to localize the string resources that PolicyCenter displays in the application user interface. These resources include display keys, typecodes, Gosu error messages, product model string resources, and workflow step names.

Note: Ruleset names and descriptions are not localized as strings. See “Localizing rule set names and descriptions” on page 45.

See also

- “Localizing workflow” on page 53

Overview of string resources

PolicyCenter uses 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 and localize them separately from other application resources.

Display keys and localization

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 key/value pairs *display keys*. You specify the key/value pair of a display key in standard Java property syntax. For example:

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

PolicyCenter stores the key/value pairs for display keys in display properties files. In the base configuration, PolicyCenter provides the file `display.properties` that defines string resources in United States English. In addition, there are multiple language files, like `display_de.properties`, `display_fr.properties`, and `display_ja.properties`.

In Guidewire Studio, you can see these files by navigating in the **Project** window to **configuration→config→Localizations→Resource Bundle 'display'**.

See also

- “Localizing display keys” on page 35
- “Enabling display languages” on page 21

Typecodes and localization

PolicyCenter displays choices and choice descriptions in drop-down lists in the user interface by using typelists that contain typecode definitions. Guidewire stores the names and descriptions of typecodes as *key/value* pairs in typelist properties files.

In the base configuration, the U.S. English string resources for typecodes are provided in the file `typelist.properties`. In addition, there are multiple language files, like `typelist_de.properties`, `typelist_fr.properties`, and `typelist_ja.properties`.

To see these files, open Guidewire Studio™ and navigate in the **Project** window to **configuration→config→Localizations→Resource Bundle 'typelist'**.

You specify the key/value pairs for the name and description of a typecode in standard Java property syntax. For example:

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

See also

- “Localizing typecodes” on page 37
- “Enabling display languages” on page 21

Workflow step names and localization

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 workflow” on page 53
- “Localizing workflow step names” on page 54

Exporting and importing string resources

PolicyCenter enables you to export some string resources to an external file, such as display keys and typecodes.

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.

```
gwb exportLocalizations [-Dexport.file] [-Dexport.language]
gwb importLocalizations [-Dimport.file] [-Dimport.language]
```

The commands provide parameters that you can use 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 property files.

See also

- “Exporting localized string resources with the command-prompt tool” on page 33
- “Importing localized string resources with the command-prompt tool” on page 33
- “Localizing string resources by exporting and importing files” on page 34

Exporting localized string resources with the command-prompt tool

PolicyCenter provides a command-prompt tool to manually export certain string resources. 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, ensure that the application server is running and navigate to your application installation directory. Then run the following command:

`gwb exportLocalizations -Dexport.file=` *`targetFile`* `-Dexport.language=`*`languageCode`*

- Command-prompt 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.
- Command-prompt parameter `-Dexport.language` specifies *`languageCode`*, the language code that PolicyCenter uses to determine the files from which to extract the resource strings. The language name must match a PolicyCenter language type that exists in the `LanguageType` typelist. For example, specifying `ja_JP` indicates that PolicyCenter is to look for property files that have that language code in the file name, such as `display_ja_JP.properties`.

See also

- “Importing localized string resources with the command-prompt tool” on page 33
- “Overview of string resources” on page 31

Importing localized string resources with the command-prompt tool

PolicyCenter provides a command-prompt 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, ensure that the application server is running and navigate to your application installation directory. Then run the following command:

`gwb importLocalizations -Dimport.file=` *`sourceFile`* `-Dimport.language=` *`languageCode`*

- Command-prompt 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 import directory as follows:
  - To use the import translation file that is in the default location, enter only the name of the file to import.
  - To use the translation file that is in a different location, enter either an absolute path or a relative path to the file from the root of the installation directory.
- Command-prompt parameter `-Dimport.language` specifies *`languageCode`*, the suffix to use in the names of the property files that PolicyCenter writes to the localization folder. The language name must match a PolicyCenter language type that exists in the `LanguageType` typelist, such as `fr_FR` or `ja_JP`. For example, specifying `ja_JP` indicates that PolicyCenter is to write property files that have that language code in the file name, such as `display_ja_JP.properties`.

See also

- "Exporting localized string resources with the command-prompt tool" on page 33
- "Overview of string resources" on page 31

Localizing string resources by exporting and importing files

You can use the `gwb exportLocalizations` and `gwb importLocalizations` commands to work with a single file of translatable strings.

The export command has the following syntax:

```
gwb exportLocalizations -Dexport.file=targetFile  -Dexport.language=languageCode
```

The exported file has all the string resources, such as display keys, in it, which cannot be guaranteed for any given property file. The string resources are separated into translated and untranslated strings, which makes it easier to see which ones you need to translate. The translated stings are extracted from the existing property files.

For example, at a command prompt, enter the following command:

```
gwb exportLocalizations -Dexport.file=my-test-en_US.txt -Dexport.language=en_US
```

This command exports results to the file `my-test-en_US.txt` in the main PolicyCenter directory.

The following code example shows some of the string resources in the untranslated and translated sections of an exported file:

```
#untranslated keys
Workflow.MetroReportWorkflow1.CheckHasReportDocumentReady.Step=CheckHasReportDocumentReady
Workflow.MetroReportWorkflow1.CheckOnInquiry.Step=CheckOnInquiry
Workflow.MetroReportWorkflow1.CheckOnOrder.Step=CheckOnOrder
Workflow.MetroReportWorkflow1.DownloadClosedReport.Step=DownloadClosedReport
Workflow.MetroReportWorkflow1.DownloadReport.Step=DownloadReport
...
#already translated
Admin.Workload.WorkloadClassification.General=General
AdminData.ActivityPattern.Description.Account_Denial=Review denial decision with Account Manager
AdminData.ActivityPattern.Description.Account_Fatality=Consult Account regarding fatality
AdminData.ActivityPattern.Description.Account_Instructions=Review all Special Handling instructions
AdminData.ActivityPattern.Description.Account_Matter=Review matter-related Special Handling instructions
AdminData.ActivityPattern.Description.Account_Negotiation=Review negotiation strategy with Account
...
```

Because the export was done for `en_US`, all the display keys, typekey names, and so on are in U.S. English.

After you make your translations, import the translated file.

The import command has the following syntax:

```
gwb importLocalizations -Dimport.file=sourceFile  -Dimport.language=languageCode
```

All strings in the file that you import, whether translated or not, are saved in property files with that extension. PolicyCenter extracts the stings for the appropriate file and updates that file. For example, typekey strings are extracted and stored in `typelist_`*`languageCode`*`.properties` file for that locale code.

See also

- "Exporting localized string resources with the command-prompt tool" on page 33
- "Importing localized string resources with the command-prompt tool" on page 33

Localizing display keys

PolicyCenter initializes the display key system as it scans for the display key property files. These files are named `display.properties` for the fallback language (U.S. English in the base configuration) and `display_languageCode.properties` for additional languages. For example, display keys in German are in the file `display_de.properties`.

You can see these files in Guidewire Studio™ by navigating to **configuration→config→Localizations→Resource Bundle 'display'**.

This node contains display key property files for each of the languages supported by Guidewire.

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

Translation technique	Related topic
Using the Studio **Display Keys** editor	"Localize display keys by using the Display Key editor" on page 36
Using the display key import and export tools	"Exporting and importing string resources" on page 32

Many of the display keys you localize are string values that PolicyCenter displays as labels or messages. There are no restrictions on the translated text for these types of display keys.

You can also localize QuickJump commands in display key property files. When you localize a QuickJump command, there must not be any spaces in the translated text. For example, the following base configuration display key in `display.properties` must be translated into a term with no spaces:

```
Web.QuickJump.RunBatchProcess = RunBatchProcess
```

See also

- "Display keys and localization" on page 31
- *Configuration Guide*
- *Application Guide*

PolicyCenter and the master list of display keys

On startup, PolicyCenter uses the display property files to generate 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:

- The master list does not already contain the display key.
- The master list already contains the display key, but, the display key in the master list has a different number of arguments from the display key being added. In this case, PolicyCenter logs a warning message noting that it found a display key value with different argument lists in different locales. For example:

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

As PolicyCenter creates the master display key list, it scans for `display.properties` and `display_languageCode.properties` files in the following order:

1. The application primary language code properties file, as set by configuration parameter `DefaultApplicationLanguage`
2. All other language code properties files configured for use by the server

3. The fallback language code properties file, which in the base configuration is `display.properties`
4. All remaining language code properties files

After PolicyCenter creates the master list of display keys, the application checks the display keys for the default language against the master list. PolicyCenter then logs as errors any display keys that are in the master list but are missing from the primary application language. 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 display key property file.

Localize display keys by using the Display Key editor

About this task

It is possible to enter a localized version of a display key directly in the Guidewire Studio editor.

Note: 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 32.

Procedure

1. Open Guidewire Studio.
2. In the **Project** window, navigate to the following location and double-click `display.properties` to open this file in the editor:

   configuration→config→Localizations→Resource Bundle 'display'
3. Place the cursor on the display key line that you want to edit.
4. Click the orange dot that appears at the beginning of the selected line and click **Edit Display Key**.
5. Modify the text for the language that you want to modify in the appropriate place in the **Edit Display Key** dialog.

   The editor saves the text that you enter for a specific language in the appropriatly named `display_`*`LanguageCode`*`.properties` file.

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 PolicyCenter installation directory:

```
gwb diffDisplayKeys
```

If necessary, the `diffDisplayKeys` tool creates a `build/missing-display-keys/config/locale` directory under the PolicyCenter installation directory, where it stores the generated diff files.

If the tool detects that a language has missing display keys:

- The tool creates a `display_`*`LanguageCode`*`.properties` file, using the language code for that language to name the file.
- The tool populates the file with the list of missing keys.

Each `display_`*`LanguageCode`*`.properties` file contains a list of display keys that are in the master list but not in the localization files. 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
```

```
#
Java.Validation.Messages.SIMPLE.SimpleGroupHeaderLabel = Es befinden sich {0} Meldungen auf Seite {1}
Java.Validation.Messages.SIMPLE.SimpleGroupHeaderLabelSingleMessage = Es befindet sich eine Meldung auf Seite {0}
Web.ClaimSummaryDV.Title = {Term.LossDetails.Proper}
```

Note: PolicyCenter does not generate a `display_*LanguageCode*.properties` file for a language 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 language code properties file that is not actually localized yet. This display key is simply a placeholder string for a display key that you intend to translate 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.

Localizing typecodes

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

Using the gwb export and import commands

Use these commands to translate all typecodes by editing a single text file. See “Localizing string resources by exporting and importing files” on page 34.

Using the Typelist Localization editor

Enter translated values for individual typecodes directly through the Typelist editor. See “Localize typecodes in a typelist properties file” on page 37.

Editing the `typelist_*LanguageCode*.properties` file for the associated language

Navigate in the Guidewire Studio **Project** window to **configuration→config→Localizations→Resource Bundle 'typelist'** and open the file so that you can edit it.

Note: In the base configuration, the file `typelist.properties` contains the U.S. English typekey name definitions.

Localize typecodes in a typelist properties file

About this task

You can use the Localization editor in Guidewire Studio™ to translate resource strings for typekeys.

Procedure

1. Navigate in the Studio **Project** window to **configuration→config→Localizations→Resource Bundle 'typelist'**.
2. Open the typelist properties file for the language and typelist that you want to localize.

   For example, double-click `typelist_de.properties`.
3. Enter `TypeKey.*Typelist*.*Typecode*` to provide a localized version of a typecode name.

   For example, enter the following for the typekey name field:

   ```
   TypeKey.PhoneType.Cell = Mobil
   ```
4. Enter `TypeKeyDescription.*Typelist*.*Typecode*` to provide a localized version of a typecode description.

   For example, enter the following for the typekey description field:

   ```
   TypeKeyDescription.PhoneType.Cell = Mobil
   ```

See also

- “Localizing string resources by exporting and importing files” on page 34

Setting a localized sort order for localized typecodes

You can set the sort order for typecodes in a typelist for specific languages in Guidewire Studio.

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

Set the sort order for typecodes in a typelist

About this task

You set the sort order in a `.sort` file. PolicyCenter does not provide any sort order files in the base configuration. You must put any `.sort` file that you create in the **Localizations** folder, as shown in this example.

Procedure

1. Create a file that has the name of the typelist and locale code with a file extension of `.sort`.

   For example, `State_de.sort`.
2. Save the file in **configuration→config→Localizations**.

   PolicyCenter stores the sort order information by language in the typelist table.
3. In the file, list the typecode display names in the appropriate language, one per line, in the order in which you want them to appear.

   Lines that are not typecode display names or that are duplicate typecode display names are reported as errors in the log file. You can also add comments on separate lines. Each comment line must start with two slash characters, //.

Sort order, typecode order, and typekey priority

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 prefectures, which customarily are in order from north to south—Hokkaido, Aomori, Iwate, so on.

Typically, instead of using `.sort` files, typecode order is determined by the priority defined in the typelist and by the sort order of the typecode display name. If a `.sort` file is present for a language, then the typekey priority is not used to determine sort order for that language. See “Determining the order of typekeys” on page 140.

Sort prefectures in alphabetic order

About this task

In the base configuration, the `State` typelist uses typekey priority to make the Japanese prefectures appear in the north-to-south order. Hokkaido has priority 1, Aomori has priority 2, Iwate has priority 3, and so forth. However, English-speaking, non-Japanese users might expect to see the prefectures listed in alphabetic order.

Procedure

1. Add a `State_ja_JP.sort` file in **configuration→config→Localizations**.
2. In that file, list the prefectures in Japanese and in north-to-south order.
3. Then do either of the following:
   - Remove the priorities for the prefectures from `State.ttx` file.
   - Create an empty `State_en_US.sort` file in the **Localizations** folder.

Example of state typelist sort file elements

The following example is not a likely one and is included only to demonstrate the elements of the file. It applies to typecodes from the `State` typelist. The file causes some U.S. western states to be listed in the order specified, and first in any U.S English list of states in the PolicyCenter user interface. The other lines in the file do not affect the list of states because they start with //. The file is named `State_en_US.sort` and is intended for U.S. English:

```
// sort order definition for some U.S. western states
Washington
Oregon
California
Nevada
Utah
Arizona
// Other states will be listed in
// alphabetical order after Arizona.
```

Any typecodes in the typelist that are not in the `.sort` file are listed after the typecode display names listed in the `.sort` file. These typecodes are ordered according to the sort order specified in the `language_`*`languageCode`*`.xml` file for that language.

Accessing localized typekeys from Gosu

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

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

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

```
var displayString = 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 as a method 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 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 property files.

Guidewire stores these string resources in the files `productmodel.display.properties` and `productmodel.display_`*`languageCode`*`.properties`. For all languages in the base configuration, Guidewire provides translated versions of the following:

- Product model Name and Description fields
- Question Name and Description fields
- Question Text and Failure message fields

Guidewire provides the U.S. English version of these fields in `productmodel.display.properties`.

To access these files, navigate in the Studio **Project** window to **configuration→config→Localizations→Resource Bundle ('productmodel.display')**.

See "Localizing PolicyCenter string resources" on page 31 for details of working with display string property files.

Translating product model strings in Product Designer

To facilitate translating product model strings in Product Designer, use the **Display Key Values by Language** 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 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.

chapter 5

Localizing PCF fields

There are a number of localizations of PCF fields that you can perform. They are dependent on the display language being used by PolicyCenter.

See also

- "Selecting language and regional formats in PolicyCenter" on page 14

Setting the default width for input field labels

The following PolicyCenter installation file defines the default widths to use for PCF field labels in various languages:

`6_platform_i18n.scss`

This file exists in the following location in Guidewire Studio for PolicyCenter:

configuration→ webresources→sass→themes→global_variables→2_platform_globals

This file is read-only in Guidewire Studio. To change these values, modify the theme to override these variables.

See also

- For information on modifying a theme, see the *Configuration Guide*.

Localizing hints for date and time fields

You can localize hints for date, time, and date-time fields in PolicyCenter screens so they can change based on the language currently in use in the application. *Hints* display as temporary text in the field and as a tooltip. The hint disappears when the user starts to enter data.

You use Guidewire Studio to localize regional format patterns for date-time field display and input in `localization_localeCode.xml` files. In these files, you can specify values for date and time formats in the `<GWLocale>` element by defining settings for `<DateFormat>` and `<TimeFormat>`.

PolicyCenter can display a hint for a date-time field based on the format values it finds in the current region's `localization_localeCode.xml` file. You can additionally localize date and time hints by language, enabling the hints to display letters that reflect the current language selection. You do this localization in `language_languageCode.xml` files.

You define and manage language characteristics in `language_languageCode.xml` files. You can define a `language_languageCode.xml` file for each language you want to support, such as `language_en_US.xml`. You access the localization files in Studio by navigating in the **Project** window to **configuration→config→Localizations**.

Note: Because you cannot define hints for Japanese characters—katakana and hiragana—the Japanese Imperial Calendar does not support configuring date-time hints.

Each `language_languageCode.xml` file contains a `<GWLanguage>` element. This element supports the following elements that you can use to configure and localize date-time field hints:

- `<FormatPatternLocalization>` – The XML subelement of `<GWLanguage>` for this localization.
- `useExampleDate` – An optional boolean attribute of `<FormatPatternLocalization>` that specifies whether the hint for a date-time field is to use a sample date-time element, with numbers, or a format pattern. By default, this attribute is `false`, and the hint for a date-time field uses a date-time pattern format, such as `MM/dd/yyyy hh:mm:ss`. If `useExampleDate` is `true`, the date-time field hint uses numbers instead of the pattern format. In either case, the hint uses the `<DateFormat>` and `<TimeFormat>` pattern definitions if they are defined in the region's `localization_localeCode.xml` file.

  For example, the following line of XML code defines `useExampleDate` attribute to be true:

  ```
  <FormatPatternLocalization useExampleDate="true"/>
  ```

- `<PatternSymbolLocalization>` – A subelement of `<FormatPatternLocalization>` that maps the letters used for the Java date-time format pattern from the Java values to the localized value.
  - `java` – The value of the Java string to be localized.
  - `localized` – The localized value of the Java string. Because this value is a string, it is possible to define a localized value for a Java pattern letter that is more than one letter.

  For example, the following code maps the Java `d` format pattern letter (the day part of a date) to French `j` (for jour):

  ```
  <FormatPatternLocalization>
    <PatternSymbolLocalization java="d" localized="j"/>
  </FormatPatternLocalization>
  ```

See also

- For information on configuring date and time fields for a region, see "<GWLocale> XML element of a localization file" on page 69.
- For information on configuring languages, see "Working with languages" on page 19.

chapter 6

Working with a localized Guidewire Studio

Guidewire Studio is built on JetBrains IntelliJ IDEA. JetBrains does not support localization of IntelliJ. However, Guidewire does provide, limited localization support for the Guidewire plugins that comprise Guidewire Studio in IntelliJ. Guidewire provides translations for the Studio editors for a limited number of languages only.

To support viewing Guidewire Studio editors in a different language, Guidewire provides the following:

- Guidewire application language files for specific languages
- Guidewire Studio language files for specific languages

Application language files provide translations for application resources such as field labels in the user interface. You also see some of these translated elements in the Studio PCF editor. Additionally, the application language files provide translation for Gosu error messages in Studio.

In particular, in the base configuration:

- Guidewire provides a `display.properties` file and a `typelist.properties` file that provide U.S. English definitions of strings in elements that you edit in Studio editors.
- Guidewire provides `display_`*`languageCode`*`.properties` and `typelist_`*`languageCode`*`.properties` files that provide translations of strings in elements that you edit in Studio editors..
- Guidewire provides internal property files in JAR files that translate strings in Guidewire Studio editors, such as labels for user interface elements. These files support Japanese.

If you enable a language in addition to U.S. English, the `display.properties` or display_*languageCode*.properties files affect PCF files that you open in the PCF editor. In those files, you see translated strings for all the PCF elements that use display keys, such as labels, buttons, and so on.

The internal display property files for Studio files provide translations for some of the labels and controls in the Guidewire editors. Not all labels and controls can be translated due to the limitations of IntelliJ. Therefore, it is possible to see a mixture of English and non-English labels and controls in the Studio user interface.

See also

- “Specifying a language for Studio” on page 44
- “Enabling display languages” on page 21

Specifying a language for Studio

You can specify a display language for Guidewire Studio either from the command line or in the **Settings** dialog after Studio starts up. As described in the previous topic, specifying a display language affects only certain aspects of the Guidewire Studio menus and windows.

Specify a language for Studio in the Settings dialog

About this task

In Guidewire Studio, you can change to a different display language if the language is supported by Guidewire.

Note: You can see the supported languages by navigating in the **Project** window to **configuration→config→Localizations→Resource Bundle ('display')**.

Procedure

1. In Studio, navigate to **File→Settings→Guidewire Studio**.
2. At the top of the **Guidewire Studio** settings page under **Language Settings**, click **Language**.
3. In the drop-down list that opens, choose either Japanese or Chinese to see the full set of user interface translations.
4. If you open a PCF file in the editor, its labels, button names, and other strings that use display keys in are in the language you chose. If you chose a language other than Japanese or Chinese, all other Studio messages, labels, menus, and so on continue to be in English.

See also

- "Enabling display languages" on page 21

Viewing Unicode characters in Studio

You can define Unicode characters in elements of Guidewire Studio and then set a theme in your operating system so you can see Unicode characters.

Elements of Studio that support viewing Unicode

You can define and view Unicode characters for the following elements in Guidewire Studio™:

- Database column descriptions, but not names
- Gosu identifiers, such as class names, method names, and variable names
- Gosu comments
- Typekey/Typecode names and descriptions, but not codes
- XSD element and attribute names
- Web service method names

You cannot define Unicode values for the following elements visible in Studio:

- Database column names, which are limited to a-z, A-Z, 0-9, and underscore
- PCF file names or the ID property
- Typekey and Typecode codes

Set Studio to view Unicode characters

Before you begin

Guidewire does not supply fonts for your system. You must download the fonts for the languages that you want to use in Guidewire Studio. Microsoft Windows has downloadable support for many languages. For example, if you

install support for Korean, Chinese, or Japanese, the downloaded files include Unicode character support for the characters used by those languages. Linux might require special font downloads.

About this task

To see Unicode characters in Studio, set a theme that is appropriate for your operating system.

Procedure

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**.

Localized Gosu error messages

Guidewire provides localized Gosu error messages in read-only property files that are stored in JAR files. You cannot edit these files. The languages supported are U.S. English and Japanese.

Localizing rule set names and descriptions

In Guidewire 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 Guidewire Studio™, to access the rule sets, navigate in the **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/...
```

Following is a description of the 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 Guidewire 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 a language for a block of Gosu code

You can set a specific language in any Gosu code block by wrapping the Gosu code in any of the following methods.

```
// First method - Sets language only
gw.api.util.LocaleUtil.runAsCurrentLanguage(alternateLanguage, \ -> { code } )

// Second method - Sets both language and region
gw.api.util.LocaleUtil.runAsCurrentLocaleAndLanguage(alternateLocale, alternateLanguage, \ -> { code } )
```

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

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.

Method parameters

The two listed methods use some, or all, of the following parameters.

`alternateLocale`

An object of type `ILocale` that represents a regional format from the `LocaleType` typelist. Specify a `GWLocale` object for this parameter.

`alternateLanguage`

An object of type `ILocale` that represents a language from the `LanguageType` typelist. Specify a `GWLanguage` object for this parameter.

`\ -> { code }`

A Gosu block as a `GWRunnable` 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 `GWLangauge` for the first parameter to `runAsCurrentLangauge`. 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 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 Guidewire Studio™ to be able to use it in `gw.i18n.ILocale`.

IMPORTANT During development, if you make a change that requires that a language typecode in the `LanguageType` typelist not be in effect when you restart, you must retire that language typecode. Do not delete the language typecode, or you might have database errors when you restart the application.

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
- “Add a locale code to the LocaleType typelist” on page 68
- For information on Gosu blocks, see the *Gosu Reference Guide*

Methods on gw.api.util.LocaleUtil

In addition to `runAsCurrentLanguage`, `runAsCurrentLocaleAndLanguage`, and `runAsCurrentLocale`, the class `gw.api.util.LocaleUtil` provides a number of other methods useful in working with localization, including the following:

Method	Description
`canSwitchLanguage`	Returns boolean `true` if the current user is assigned to a role that has the `usereditlang` permission, which allows the user to switch to a different language.
`canSwitchLocale`	Returns boolean `true` if the current user is assigned to a role that has the `usereditlang` permission, which allows the user to switch to a different locale.
`getAllLanguages`	Returns a list of the typecodes of all languages defined in the `LanguageType` typelist.
`getAllLocales`	Returns a list of the typecodes of all locales defined in the `LocaleType` typelist.
`getCurrentLanguage`	Returns the effective language for the thread as a `GWLanguage` object. This language can be, in order of priority, the temporary setting, such as by `runAsCurrentLanguage`, the user setting, or the system setting.
`getCurrentLocale`	Returns the effective locale for the thread as a `GWLocale` object. This locale can be, in order of priority, the temporary setting, such as by `runAsCurrentLocale`, the user setting, or the system setting.
`getCurrentLanguageType`	Calls `getCurrentLanguage` and returns a `LanguageType` typecode.
`getCurrentLocaleType`	Calls `getCurrentLocale` and returns a `LocaleType` typecode.
`getDefaultLanguage`	The system setting for the language, set in configuration parameter `DefaultApplicationLanguage`.
`getDefaultLocale`	The system setting for the locale, set in configuration parameter `DefaultApplicationLocale`.
`getLanguageLabel`	Returns the localized display name for the language as a `String`. For example, if the current language is English and the method parameter is `LanguageType.TC_DE_DE`, the method returns `"German"`. However, if the current language is German, the method returns `"Deutsch"`.
`getLocaleLabel`	Returns the localized display name for the locale as a `String`. For example, if the current language is English and the method parameter is `LocaleType.TC_DE_DE`, the method returns `"German (Germany)"`. However, if the current language is German, the method returns `"Deutsch (Deutschland)"`.
`toLanguage`	Converts a `LanguageType` typekey to a `GWLanguage` object and returns it as an `ILocale` object.
`toLocale`	Converts a `LocaleType` typekey to a `GWLocale` object and returns it as an `ILocale` object.

chapter 7

Localizing administrative data

Guidewire refers to certain types of application data as *administration data*. For example, activity patterns are administrative data. For selected fields in administrative data, PolicyCenter stores localized values directly in the application database.

You enter translations for administration data directly in PolicyCenter 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 at the bottom of a screen. This list view has a row for each enabled language in the application. Each row has fields showing which elements on that screen you can localize.

Specifying localized columns in entities

To accommodate localized values for shared administrative 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. You select the entity and then right-click the column you want to localize and choose **Add new→localization**.

Adding a `localization` element to a column causes a localization table to be created 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 50.

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
`nullok`	`Boolean`	This attribute is required. Set it to the same value as the `nullok` attribute for the column to which you are adding the table.

Attribute	Type	Description
		If you set this attribute to `false`, 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: • PolicyCenter stores the values for the default application language in the main database table of the entity. • PolicyCenter stores the values for additional languages in a separate localization table. 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. **Note:** If only one language is configured, PolicyCenter does not run the consistency check.
`tableName`	`String`	The name of the localization table. Use the following format for this name: *`mainEntityNameAbbrevation_columnNameAbbreviation`*`_l10n` **IMPORTANT:** The table name must be no longer than 16 characters. If the name exceeds this length, the application server will not start.
`extractable`	`Boolean`	Default value is `false`. If you set this attribute to `true`, PolicyCenter adds the localization table to the archive for the entity. See the *Configuration Guide*.
`overlapTable`	`Boolean`	Default value is `false`. 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 `true`, PolicyCenter marks the localization table as an overlap table. See the *Configuration Guide*.
`unique`	`Boolean`	Default value is `false`. If you set this attribute to `true`, PolicyCenter enforces that for each language the values are unique and there are no duplicates. If the entity is of type `effdated` or `effdatedbranch`, do not set `unique` to `true`. See the *Configuration Guide*.

See also

- *System Administration Guide*

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.

```
mainEntityNameAbbrevation_columnNameAbbreviation_l10n
```

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_l10n
```

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
- Underwriting companies

See also

- For complete information on system tables, see the *Product Model Guide*.

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. Additionally, you can directly edit system table display properties files in Studio.

See also

- For information on configuring a column as a localized column, see “Specifying localized columns in entities” on page 49.
- For information on how the database stores localized values, see “Localization tables in the database” on page 50.

Localizable tables in PolicyCenter

The default configuration of PolicyCenter supports localizing the following system table columns.

System table file	System table entity	Localized columns
`bop_class_codes.xml`	`BOPClassCode`	`Classification`
`class_code_basis.xml`	`ClassCodeBasis`	`Name, Description`
`cp_class_codes.xml`	`CPClassCode`	`Classification`

System table file	System table entity	Localized columns
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	WCClassCode	Classification, ShortDesc

Localizing system table XML files

Guidewire recommends that you use the Product Designer system table editor to make changes to system tables. However, if you need to make numerous changes for localization, you can modify the system table files directly. System tables are in XML format.

The localization elements in each system table have a similar XML structure. In the system table XML file, elements that can be localized use a `language` attribute to provides translated text in for that system table.

In the following `CPClassCode` system table example, the `language` attribute of the `Classification` element is used to provide German, U.S. English, Spanish, French, Italian, Dutch, Portuguese, and Russian text for the `Classification` field.

```
<CPClassCode public-id="CP_0082_5">
    <ClassIndicator>5</ClassIndicator>
    <Classification language="de">Klärwerke - Pumpenhäuser, Chloranlagen, Faultürme, Filter usw.</Classification>
    <Classification language="en_US">Sewage Treatment Plants - pump houses, chlorinators, digesters, filters, et</
Classification>
    <Classification language="es">Plantas de tratamiento de aguas residuales: estaciones de bombeo, cloradores,
digestores, filtros, etc.</Classification>
    <Classification language="fr">Usine de traitement des eaux usées : pompes, électrolyseurs, digesteurs, filtres,
etc.</Classification>
    <Classification language="it">Impianti di trattamento delle acque reflue - locali pompe, cloratori, digestori,
filtri ecc.</Classification>
    <Classification language="nl">Afvalwaterzuiveringsinstallaties: pomphuizen, chloreertoestellen, gistingstanks,
filters, enzovoort</Classification>
    <Classification language="pt">Usinas de tratamento de esgotos - casas de bomba, cloradores, digestores, filtros
etc.</Classification>
    <Classification language="ru">Канализационные очистные сооружения – здания насосной станции, хлораторные
установки, утилизационные котлы, фильтры, et</Classification>
    <Code>0082</Code>
    <EffectiveDate>2000-01-01 00:00:00.000</EffectiveDate>
    <ExpirationDate />
</CPClassCode>
```

Note: In the base configuration, there are also translations for Japanese and Chinese that are not shown in the preceding example.

chapter 8

Localizing 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 59 for information on localizing application documents, notes, and emails.

Set the workflow language or region

About this task

It is possible to set the workflow region and the workflow language independently of the default application language and region in Guidewire Studio™.

Procedure

1. Open Guidewire Studio for PolicyCenter.
2. In the Studio **Project** window, navigate to the following location and double-click the workflow node of interest to open it in the Workflow editor:

   configuration→config→Workflows
3. Click the background area in the workflow layout view.

   This action opens the **Properties** area at the bottom of the workflow area.
4. In this **Properties** area, you can enter one of the following:
   - A fixed name for the language or region
   - A Gosu expression that evaluates to a valid type for the language or region

   For example:

Type	Gosu
Fixed string	`gw.i18n.ILocale.EN_US`
Variable expression	`gw.api.util.LocaleUtil.toLanguage(PolicyPeriod.Policy.PrimaryLanguage)` `gw.api.util.LocaleUtil.getCurrentUserLanguage()`

Localizing workflow step names

Guidewire provides translated versions of workflow step names. PolicyCenter displays translated step names in the following screens.

Workflows screen

On the **Administration→Monitoring→Workflows** search screen, the workflow summary shows the last completed workflow step. Only administrative accounts can access this screen.

Workflow Detail screen

On the **Workflows** screen, select a workflow to view the log for that workflow that shows all the workflow steps.

Additionally, Guidewire Studio can show translated step names in the Workflow editor if you specify the translated language as you start Studio.

There are two techniques you can use to translate workflow step names:

- To translate a small number of names, see "Translate workflow step names in Studio" on page 54.
- To translate a large number of names, see "Export workflow step names as string resources for translation" on page 55.

See also

- "Selecting language and regional formats in PolicyCenter" on page 14
- "Specifying a language for Studio" on page 44

Translate workflow step names in Studio

About this task

Guidewire provides translated step names for a number of languages. You can translate workflow step names directly in the Guidewire Studio™ Workflow editor. This technique is useful if you want to translate a small number of names.

Procedure

1. Open Guidewire Studio for PolicyCenter.
2. In the Studio **Project** window, navigate to the following location and double-click the name of workflow that you want to open:

   configuration→config→Workflows
3. Click the **Text** tab at the bottom of the editor to show the XML for the workflow.
4. Each translation of a workflow name is in a `<StepLocalization>` element.
5. Find the element for the locale you want to translate.

   For example, for a German translation, look for:

   ```
   locale= "de"
   ```
6. Enter the translation for the workflow name in the `name` element.

   For example, for the U.S. English name "Pending Step", enter:

   ```
   name= "Ausstehende Stornierung"
   ```

See also

- "Specifying a language for Studio" on page 44

Export workflow step names as string resources for translation

About this task

If you have more than a few workflow step names to translate, you can export them, translate them in the exported file, and then import them.

Procedure

1. Export the PolicyCenter string resources for a particular locale by using the following command in the installation directory:

   ```
   gwb exportLocalizations
          -Dexport.file=targetFile
          -Dexport.language=languageCode
   ```

   In the command, you must provide a target file name and specify the language code to use to identify the files from which to export the string resources.
2. In the exported file, find the workflow by display key name.

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

   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 example represents that character as a period.
3. Find each `<StepLocalization>` element for the language you want to translate.
4. Translate each workflow step name into the target language.
5. Import the translated strings resources back into PolicyCenter by using the following command syntax:

   ```
   gwb importLocalizations
          -Dimport.file=sourceFile
          -Dimport.language=languageCode
   ```

   In the command, you must provide a source file name and specify the language code to use in the file name that the command saves in the localization folder.

See also

- “Exporting and importing string resources” on page 32
- “Localizing display keys” on page 35

Creating a language-specific workflow subflow

You can use subflows to implement simple parallelism in internal workflows, which is otherwise impossible because a single workflow instance cannot be in two steps simultaneously. One use of a subflow is to make it language specific.

Methods that create a language-specific subflow

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

Method	Description
`createSubFlow`	Creates a child subflow synchronously. PolicyCenter starts the subflow immediately upon method invocation. The new subflow automatically uses the default application language, not the language of the parent workflow. Thus, if you set the language of the parent workflow to be different from the default application language, the subflow does not inherit that language.
`createSubFlowAsynchronously`	Creates a child subflow that PolicyCenter starts only after executing all code in the Gosu initialization block. The subflow uses the default application language, not the language 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 language of the subflow before the workflow starts.
`instantiateSubFlow`	Creates a child subflow, but does not start it. You can modify the subflow that the method returns before you start the subflow.

Create a child subflow

About this task

You can create a child subflow that inherits the language of the parent workflow. This example uses the `Workflow` method `instantiateSubflow`. There are other methods you can use, as described in "Methods that create a language-specific subflow" on page 55.

Procedure

1. Define a workflow that has the `LanguageType` property.

   See the *Configuration Guide* for information on how to create a new workflow with a `LanguageType` property.
2. Set the language for this subflow so that it uses your preferred language.
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 language of the parent workflow.
5. Start the subflow by using one of the workflow start methods described in the *Configuration Guide* topic on instantiating a workflow.

See also

- *Configuration Guide*

Localize Gosu code in a workflow step

About this task

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.

Note: Besides `runAsCurrentLanguage`, other `LocaleUtil` wrapper methods useful for localization include `runAsCurrentLocale` and `runAsCurrentLocaleAndLanguage`.

Procedure

1. Wrap the Gosu code in the following method:

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

2. 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 } )
```

See also

- “Setting a language for a block of Gosu code” on page 45
- “Setting regional formats for a block of Gosu code” on page 75

chapter 9

Localizing templates

In the base configuration, Guidewire provides a number of template-related definition files for notes, emails, and documents. In Guidewire Studio™, you can navigate in the **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`	Template descriptor file in XML format. This file contains the template metadata, such as: • `name` • `subject` This file can also contain symbol definitions for context objects that PolicyCenter substitutes into the template content file in creating the final document.	`CreateEmailSent.gosu.htm.descriptor`

See also

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

- *Gosu Reference Guide*
- *Integration Guide*

Creating localized documents, emails, and notes

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

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

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.

To create localized version of document, email, and note templates, use the following multi-step process.

1. “Create language-specific folders” on page 60
2. “Copy template content files” on page 61
3. “Localizing template descriptor files” on page 61
4. “Localize template files” on page 63
5. “Localizing documents, emails, and notes in PolicyCenter” on page 64

Create language-specific folders

About this task

In Guidewire Studio™ for PolicyCenter, 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**

You can create your own localized versions of the template files.

Procedure

1. Open Guidewire 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 language-specific folder and click OK.

   For example, if 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**

Result

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

Example

For example:

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

Next steps

The next step is “Copy template content files” on page 61.

Copy template content files

Before you begin

Complete the step “Create language-specific folders” on page 60 before you perform this step.

About this task

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

Procedure

1. 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
   ```

2. 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
   ```

Next steps

The next step is “Localizing template descriptor files” on page 61.

Localizing template descriptor files

Complete the step “Copy template content files” on page 61 before you perform this step.

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 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” on page 62
- “Localizing email and note descriptor files” on page 63

You can continue for more localization descriptor file information or you can move to the next step “Localize template files” on page 63.

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`. To view document template descriptor files in Guidewire Studio for PolicyCenter, navigate in the Studio **Product** window to the following location:

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.

Example

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="EmailSent.gosu.htm"
      description="Record of an email being sent"
      ...
      keywords="CA, email">

  <DescriptorLocalization language="languageCode" name="localizedName"
        description="localizedDescription" />

  ...

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

  ...

</DocumentTemplateDescriptor>
```

XML elements to localize

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 you can localize in a descriptor file.

Element	Attribute	Description
`<DocumentTemplateDescriptor>`	• `keywords`	Localize the keywords associated with this template to facilitate the search for this template in the PolicyCenter search screen.
`<DescriptorLocalization>`	• `language` • `name` • `description`	Subelement of `<DocumentTemplateDescriptor>` – Enter a valid `GWLanguage` value for `language`, such as `gw.i18n.ILocale.LANGUAGE_EN_US`, which uses the language type-code `en_US`. The language typecode must be defined in the `LanguageType` typelist. See also, “Setting regional formats for a block of Gosu code” on page 75.

Element	Attribute	Description
		You can also localize the name and description of this template as it appears in PolicyCenter.
`<ContextObjectLocalization>`	• `language` • `display-name`	Subelement of `<ContextObject>` – Enter a valid `GWLanguage` value for `language`. See the previous description for more information. You can also localize the name of this template as it appears PolicyCenter.

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:

1. Place a copy of each descriptor file in the correct language folder.
2. Translate the following attributes in that file.

Element	Attribute	Description
`<emailtemplate-descriptor>` `<notetemplate-descriptor>`	• `keywords` • `subject`	Translate the keywords associated with this template to facilitate the search for this template in the PolicyCenter search screen. Also, translate the subject of this template to show that value in PolicyCenter.

For example, to localize an email or note template descriptor file for French (France):

1. First, copy the descriptor file to a `fr_FR` folder.
2. Then, provide any translated keywords that you want and the translated subject tag for the template.

Localize template files

Before you begin

Complete the step “Localizing template descriptor files” on page 61 before you perform this step.

About this task

After copying the template content files to your language folder, as described in “Copy template content files” on page 61, you then need to translate them. Unless you want to create a new template, the simplest procedure is to work with a clone of an existing template file.

Procedure

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

Next steps

The next step is “Localizing documents, emails, and notes in PolicyCenter” on page 64.

Localizing documents, emails, and notes in PolicyCenter

Complete the step "Localize template files" on page 63 before you perform this step.

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 if no language is specified. If you select one, however, PolicyCenter makes the **Language** field in the **New Document** worksheet editable.

Create a localized document

Procedure

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 (magnifying glass) in the **Document Template** field.

   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.

Create a localized email

Procedure

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.

Create a localized note

Procedure

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 these interfaces to search for document templates and generate documents by using a specific language:

- `IDocumentTemplateDescriptor` interface
- `IDocumentTemplateSource` plugin interface
- `IDocumentTemplateSerializer` plugin interface

See also

- For information on these plugins, document management, and writing and installing document templates, see the *Integration Guide*.

chapter 10

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.

Using 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_localeCode.xml` files. You save these files in the **Localizations** folder. If there is no `localization_localeCode.xml` file for a region, PolicyCenter uses the ICU Library. You configure these formatting options in Guidewire Studio™.

Configuring regional formats

Overview of 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 in the base configuration.

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 Java 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/`.

Locale codes, localization_localeCode.xml, and the ICU library

The Localizations folder does not have to contain any `localization_localeCode.xml` files. If you do add such a file, you can define only locale settings that override the ICU library defaults. For any locale settings that you have not defined, PolicyCenter uses the ICU library defaults, 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.

See also

- "Overview of the International Components for Unicode (ICU) library" on page 67

Java locale codes and the ICU library

The ICU library 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, as described in "Setting the default application locale for regional formats" on page 73
- To access the regional formats for a locale supplied by the ICU library, as described in "Overview of the International Components for Unicode (ICU) library" on page 67

Add a locale code to the LocaleType typelist

About this task

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

Procedure

1. Open Guidewire Studio for PolicyCenter.
2. In the **Project** window, navigate to the following location and right-click `LocaleType.tti`:

   configuration→config →Metadata→Typelist
3. Choose **New→Typelist extension** to create the file `LocaleType.ttx`.
   - If you see a **Typelist Extension** dialog enabling you to choose one of several `LocaleType.ttx` locations, choose the one in `configuration/config/extensions/typelist/` and then click OK.
   - If you see a message saying that typelist extension is not allowed, then the file `LocalType.ttx` already exists. Open that file instead.
4. In the editor for `LocaleType.ttx`, right-click the top element on the left, **typelistextension** `LocaleType`, and choose **Add new→typecode**.
5. For **code**, enter the Java locale code.

   For example, enter the Java locale code `nl_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_localeCode.xml file

Guidewire uses `localization_localeCode.xml` files to define regional formats. To view these files, open Guidewire Studio for PolicyCenter and navigate to the following location in the Studio **Project** window:

configuration→config→Localizations

These localization files have Java locale code suffixes, such as `localization_fr_FR.xml`.

Guidewire provides the following localization files in the base configuration:

- `localization_en_US.xml`
- `localization_fr_FR.xml`
- `localization_ja_JP.xml`

These localization files define the regional formats that are customary to each region. For example:

- The `localization_en_US.xml` file contains configuration information on date, time, number, and currency formats for use in the United States.
- The `localization_ja_JP.xml` 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_localeCode.xml` file. See "Overview of the International Components for Unicode (ICU) library" on page 67.

You can add a new localization file to the Studio **Localizations** folder. If there are multiple copies of a localization file, then the files in the main **configuration** module override any other localization files.

You can use an existing `localization_localeCode.xml` file as a starting point if you want to create a new `localization_localeCode.xml` file to override the ICU library settings for a region. You define attributes and subelements of the `<GWLocale>` element.

See also

- "<GWLocale> XML element of a localization file" on page 69

<GWLocale> XML element of a localization file

A `localization_localeCode.xml` file contains a single `<GWLocale>` 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 `<GWLocale>` element can take the following attributes and subelements.

<GWLocale> Attributes

The `<GWLocale>` element can take the following 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 following default ICU library settings 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.

<GWLocale> Subelements

`CurrencyFormat`

Currency format pattern for the region, used in single currency display mode.

`negativePattern`, `positivePattern`	Define how PolicyCenter displays positive and negative monetary amounts. PolicyCenter displays and formats the numeric value in place of the pound sign (#) in the pattern. PolicyCenter displays all other characters in the pattern as they are without modification. For example, the positive pattern `$#` displays the numeric value 32 as `$32.00`. The negative pattern `($#)` displays the numeric value -5 as `($5.00)`.
`zeroValue`	Defines how PolicyCenter displays zero amounts. For example, the zero value pattern can be `0` (zero) or `-` (dash). If the numeric value of a monetary amount is null, PolicyCenter displays the amount as empty or blank. The monetary amount must be 0.00 for the zero value pattern to be used.

`DateFormat`

Patterns for the date formatter and parser.

`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"`.

`JapaneseImperialDateFormat`

Japanese imperial calendar settings.

`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. Determines how PolicyCenter displays numbers.

`decimalSymbol`	Symbol used to separate a whole number from its decimal fraction, usually a period or a comma. For example, `"."`.
`negativeEntryPattern`	Format for entering negative numeric values. For example, `"(#)"`.
`thousandsSymbol`	Symbol used to separate groups of three digits to the left of the decimal mark, usually a period or a comma. For example, `","`.

`TimeFormat`

Time formatter and parser configuration.

`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"`.

`NameFormat`

Formatting of names by the PCF file `GlobalContactNameInputSet` or `GlobalPersonNameInputSet`.

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.

yearMonth	For example, for ja_JP, the value is "yyyyMM".

<DateFormat> and <TimeFormat> elements of a localization file

You can specify any of the following attribute values for the DateFormat and TimeFormat elements:

- short
- medium
- long

For example (for the en_US locale):

```
<DateFormat short="MM/dd/yyyy"
            medium="MMM d, yyyy"
            long="E, MMM d, yyyy" />

 <TimeFormat short="hh:mm aa"
            medium="hh:mm aa"
            long="hh:mm aa"/>
```

In general, you can use any of the date and time patterns supported by the Java class SimpleDateFormat for the medium and long formats. However, Guidewire maps the short format to the date picker widget, which does not support arbitrary date formats. For a description of these patterns, refer to the following web sites:

http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html

http://cldr.unicode.org/translation/date-time-1/date-time-patterns

PolicyCenter uses the short form to recognize dates entered by the user. It uses the other forms to display dates and times.

Using the Short format

Define patterns for the short date and time definitions that result only in a fixed-length output that matches the pattern length.

Note: Time and date inputs in PCF files do not accept variable length format patterns, such as MMM. Therefore, for these fields, use only format patterns that result in fixed-length output. If you do use a variable-length input pattern, it is coerced during input into the short form.

In general, dates have three components, year, month, and day, each defined by a specific pattern. Each pattern provides a fixed-width output. For example, the following patterns all provide a fixed-width output. In this case, each output contains the same number of characters as the format pattern.

Format	Pattern	Output
year	yyyy	4 digit output, fixed
month	MM	2 digit output, fixed
day	dd	2 digit output, fixed

A pattern that does not work

The following list describes an incorrect, non-fixed-width pattern:

Format	Pattern	Output
year	yyyy	4 digit output, fixed
month	MMM	variable length output
day	dd	2 digit output, fixed

The pattern MMM does not work as there are languages in which the abbreviated month string is not three characters in length. Attempting to use this pattern with a language in which there are abbreviated month strings that are not three characters in length can cause a validation error.

chapter 11

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 configuration file `config.xml`. In this file, set configuration parameter `DefaultApplicationLocale` to the appropriate value for your installation. In the base configuration, Guidewire sets the default application locale to `en_US`.

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 does not start.

Example

The following example sets the default application locale to `France (French)`.

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

This action sets the default choice for regional formats in Guidewire PolicyCenter.

chapter 12

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
`alternateLocale`	An object of type `ILocale` that represents a regional format from the `LocaleType` typelist. Specify a `GWLocale` object for this parameter.
`alternateLanguage`	An object of type `ILocale` that represents a language from the `LanguageType` typelist. Specify a `GWLanguage` object for this parameter.
`\ -> { code }`	A Gosu block as a `GWRunnable` 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 Guidewire Studio™ to be able to use it in gw.i18n.ILocale.

Example

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

- "Add a locale code to the LocaleType typelist" on page 68
- "Setting a language for a block of Gosu code" on page 45
- Information on Gosu blocks in the *Gosu Reference Guide*.

chapter 13

Configuring name information

Name formats shown in PolicyCenter can vary by region. For example, for `Person` contact types, as opposed to `Company` types, a contact detail view can show the following name fields:

- United States – Prefix, First Name, Last Name, Suffix
- France – Prefix, First Name, Particle, Last Name, Suffix
- Japan – Last Name (phonetic), First Name (phonetic), Last Name, First Name

  Note: For Japanese names, the labels shown on the screen map to the following columns:

  - Last Name (phonetic) maps to `LastName`.
  - First Name (phonetic) maps to `FirstName`.
  - Last Name maps to `LastNameKanji`.
  - First Name maps to `FirstNameKanji`.

You can customize this feature, as described in this topic.

Note: ContactManager supports configuration of names for globalization with features that are similar to those of PolicyCenter. Information for ContactManager name configuration is included in this topic where it differs from PolicyCenter name configuration.

Names in PolicyCenter

In PolicyCenter, a name is part of contact information, such as a policy or account owner. PolicyCenter provides modal PCF files that support name formats that vary by region.

Read-only and editable name information

PolicyCenter can display name information as read-only text or as editable text entry fields:

- PolicyCenter displays a read-only name as a display name on a single line. PolicyCenter uses the `localization_`*`localeCode`*`.xml` file for the current region and the `NameFormatter` class to determine which name fields to use.
- PolicyCenter displays editable names as a set of editable text fields in which you can add, modify, or delete information. PolicyCenter uses the `localization_`*`localeCode`*`.xml` file and the current region to determine the name fields to show.

See also

- “Configuring name data and fields for a region” on page 78

Name owners

PolicyCenter uses a `NameOwner` object to control the display and editing of names. The `NameOwner` object identifies the object that contains a particular name. The `NameOwner` determines how read-only values are formatted. For editable names, the `NameOwner` determines which PCF mode to use and which fields to show. You can define different name owners depending on your requirements.

See also

- "Modal PCF files and name configuration" on page 82

Modal name PCF files

The current region determines which name fields in the database are visible in the PolicyCenter user interface for a name. The PCF files `GlobalContactNameInputSet` and `GlobalPersonNameInputSet` have modal versions that make different sets of name fields visible based on the current region. These PCF files also control the order in which PolicyCenter displays name fields.

A single mode of these PCF files can be shared between multiple regions. You can add new fields to the existing PCF files. If you need a different set or order of name fields to display for a region, you can add a new modal PCF file for the region. You can use a copy of one of the existing modal PCF files as a starting point for the new file.

To determine which modal PCF file is used for a region, you set the `PCFMode` attribute in the `localization_localeCode.xml` file for that region.

See also

- "PCFMode attribute of the NameFormat element" on page 79.
- For more information on `GlobalContactNameInputSet` and `GlobalPersonNameInputSet`, see "Modal PCF files and name configuration" on page 82.
- For general information on modal PCF files, see the *Configuration Guide*.

Configuring name data and fields for a region

You use region-specific `localization_localeCode.xml` files to configure the data that PolicyCenter uses to display names for specific regions.

Note: For read-only name display, the order is determined by code in the `NameFormatter` class. For editable name fields, the order is determined by the order of the fields in the PCF mode for the region.

If you add a new name format for a region or you add a new `Contact` (or `ABContact`) name property, you must also configure the files that support read-only names. See "Setting up additional region and name configurations" on page 80.

Configuring the Localization XML file for names

PolicyCenter stores `localization_localeCode.xml` files in the **Localizations** folder, which you can access in Guidewire Studio™ in the **Project** window at **configuration→config→Localizations**. For example, the file for Japan is `localization_jp_JP.xml`.

File `localization_localeCode.xml` defines a `NameFormat` element that contains the following useful attributes:

Attribute	Description	More information
`PCFMode`	Specifies the mode of the appropriate PCF name file to use	"PCFMode attribute of the NameFormat element" on page 79
`textFormatMode`	Specifies the region that class `NameFormatter` uses to format a read-only name.	"Text format mode attribute of the NameFormat element" on page 79

Attribute	Description	More information
visibleFields	Specifies which name fields to display	“Visible fields attribute of the NameFormat element” on page 79

PCFMode attribute of the NameFormat element

Attribute PCFMode, on the NameFormat element in file localization_*localeCode*.xml, determines which modal version of the name input PCF files PolicyCenter uses for specific countries. The name input PCF files are GlobalContactNameInputSet and GlobalPersonNameInputSet.

For example, for Japan, localzation_jp_JP.xml file specifies the PCF mode as follows:

```
<NameFormat PCFMode="Japan" .../>
```

If the user chooses Japan as the region, PolicyCenter uses one of the following PCF files as appropriate to display name information:

- GlobalContactNameInputSet.Japan
- GlobalPersonNameInputSet.Japan

If a localization_*localeCode*.xml file does not define the PCFMode attribute, or there is no localization_*localeCode*.xml file for a region, PolicyCenter uses the default modal version of the name PCF file. In the base configuration, the default version is suitable for many countries.

Class NameLocaleSettings defines the valid values that you can use for PCFMode. If you want to add a PCF mode, you must define it in the NameLocaleSettings class. The default base configuration definition is:

```
private static final var validPCFModes : Set<String> = { "default", "Japan", "", null }
```

See also

- “Modal PCF files and name configuration” on page 82

Text format mode attribute of the NameFormat element

Attribute textFormatMode, on element NameFormat in file localization_*localeCode*.xml, specifies the region name that the NameFormatter class uses to format a read-only name. Method internalFormat pm the NameFormatter class defines the region names. The default value of the textFormatMode attribute is default.

Class NameLocaleSettings defines the value values that you can use for the textFormatMode attribute. If you want to add a text format mode, you must define it in this class. The default base configuration definition is:

```
private static final var validTextFormatModes : Set<String> =
    { "default", "France", "Japan", "", null }
```

See also

- “NameFormatter class” on page 83

Visible fields attribute of the NameFormat element

Attribute visibleFields, on element NameFormat in file localization_*localeCode*.xml, defines the set of name fields that can be visible for each region. For example:

France	visibleFields="Prefix,FirstName,MiddleName,Particle,LastName,Suffix,Name"
Japan	visibleFields="LastName,FirstName,LastNameKanji,FirstNameKanji,Name,NameKanji"

The field order and the actual fields that can be shown are specified in the PCF files or the NameFormatter class. The order of the fields defined in visibleFields does not matter, and listing a field in this attribute simply enables it to be shown if it is specified.

The fields listed in this attribute must be defined in the class `NameOwnerFieldId`. By convention, the names used in `visibleFields` match field names defined in `Contact` or a subentity of `Contact`. In ContactManager, the names match field names defined in `ABContact` or a subentity of `ABContact`. Names are case-sensitive. If the values are not valid, you get error messages in the log when the data is first used. The error messages are defined in `gw.api.name.NameLocaleSettings.init`.

If a `localization_localeCode.xml` file does define a `visibleFields` attribute, PolicyCenter uses the entire set of name fields defined in the PCF file.

See also

- "NameOwnerFieldId class" on page 85

Setting up additional region and name configurations

You can add a new name format for a region, and you can extend `Contact` or a `Contact` subtype with a new name column. These changes require that you update various files and classes.

Note: If you have ContactManager installed, you must also make corresponding changes in ContactManager.

Add a name format for a region

About this task

You can add an additional name format for a region.

Procedure

1. Configure a `localization_localeCode.xml` file.

   See "Configuring the Localization XML file for names" on page 78.
2. Define modal name input PCF files that list the input fields if the current files do not have that sequence of fields.

   See "Modal PCF files and name configuration" on page 82.
3. Add a new `if` statement to the `NameFormatter.internalFormat` method to handle the formatting of the name string for the new region.

   See "NameFormatter class" on page 83.

Extend the Contact entity with a new name column

About this task

If you extend the `Contact` entity or a subtype of `Contact` to add a new name column, you must make the following changes in PolicyCenter.

Note: In ContactManager, you would be extending the `ABContact` entity or a subtype of `ABContact` to add a new name column

Procedure

1. Add the new field to the `Contact` or the `Contact` subtype and ensure that the integration files are correctly set up.

   See the *Guidewire Contact Management Guide*.
2. Update the class `NameOwnerFieldID` to add the new field.
   a. Add a variable for the new column to this class.
   b. Add the variable as needed to any of the following existing constants that list name fields in this class:

```
ALL_PCF_FIELDS
ALL_CONTACT_PCF_FIELDS
```

```
REQUIRED_NAME_FIELDS
DISPLAY_NAME_FIELDS
FIRST_LAST_FIELDS
HIDDEN_FOR_SEARCH
```

See “NameOwnerFieldId class” on page 85.

3. Add the new field to one of the following interfaces:
   - For a non-person contact, add the field to `ContactNameFields`.
   - For a person contact, add the field to `PersonNameFields`.
4. Compile the project to see which name delegate classes you need to update.

   For name delegates that take their values from a contact or person, you will see error messages identifying those classes.
   - You will need to add the field to either `ContactNameDelegate` or `PersonNameDelegate`, depending on whether you added it to `ContactNameFields` or `PersonNameFields`. In ContactManager, these classes are `ABContactNameDelegate` and `ABPersonNameDelegate`.
   - For the remaining name delegates, decide if the new name column needs to be added to the underlying entity.
   - There are several non-persistent entities used for searching. If the new column will not be used for searching, create a getter for the field that returns `null` and a setter for the field that throws an exception. For example, `MiddleName` is not a search column in the base configuration. For an example, see the implementation of `MiddleName` in `gw.api.name.PLContactSearchNameDelegate`.
   - For persistent entities, you can add a new column to match the column you added to the `Contact` or `Contact` subtype. Then you create a getter and setter for the field in the associated name delegate. If you do not add the column to the persistent entity, create a getter for the field that returns `null` and a setter for the field that throws an exception.
5. If necessary, add a `localization_`*`localeCode`*`.xml` file for the region and then add the new field to it. Also, update the existing `localization_`*`localeCode`*`.xml` files with the field. See “Configuring the Localization XML file for names” on page 78.
6. Add the field to the appropriate modal contact input set files, `GlobalContactNameInputSet` or `GlobalPersonNameInputSet`.

   Add new fields as needed to these PCF files. For example, if the new name column applies only to persons, change only the `GlobalPersonNameInputSet` modal files as needed.

   You might not have to change the file for all modes. Configure your new fields by following the pattern of the existing fields in the file.

   See “Modal PCF files and name configuration” on page 82.
7. Use the new field in the `NameFormatter` class.

   In the `internalFormat` method, add a call to `append` for the new field. Add the `append` call to the `if` statement for the `mode` value that matches the region for the new field. Follow the pattern used for the other regions and fields. The `fieldid` values are defined in `gw.api.name.NameOwnerFieldId`. See “NameFormatter class” on page 83.
8. Find display names that use the `NameFormatter` class.

   Press `Ctrl+Shift+F` and search for `NameFormatter` with **Scope** set to **All Places** and **File mask** set to `*.en`.
   - The following list shows the display names that use `NameFormatter` in the base configuration of PolicyCenter:

     ```
     CommercialDriver.en
     Company.en
     Contact.en
     Place.en
     PolicyContactRole.en
     ```
   - The following list shows the display names that use `NameFormatter` in the base configuration of ContactManager:

```
ABCompany.en
ABContact.en
ABPlace.en
```

9. Add the new field as needed to the entity display names that use the `NameFormatter` class.

   In the Entity Names editor, add the field both to the list of fields in the table at the top of the screen and to the code in the lower text entry area.

   Note: If the new name field applies only to persons, you need to add the name field to `Contact.en` but not to `Company.en` or `Place.en`

   For example `:NewFieldName = newFieldName`.

Modal PCF files and name configuration

PolicyCenter and ContactManager use one of two modal PCF files for names:

- `GlobalContactNameInputSet` – Used for `Company` names. In ContactManager, used for `ABCompany` names.
- `GlobalPersonNameInputSet` – Used for `Person` names. In ContactManager, used for `ABPerson` names.

Modal versions of the global name PCF files determine the available fields and their order for specific countries.

In the base configuration, PolicyCenter provides the following modal versions of the global name input set files.

Modal Name PCF Files	Region
• `GlobalContactNameInputSet.default` • `GlobalPersonNameInputSet.default`	• Australia • Canada • France • Germany • Great Britain • United States
• `GlobalContactNameInputSet.Japan` • `GlobalPersonNameInputSet.Japan`	• Japan

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

Mapping regions to modes

A region maps to a mode through its associated localization folder's `localization_`*`LocaleCode`*`.xml` settings. If there is no localization file for the region, the `default` mode is used.

Controlling field properties

Each modal PCF file uses an implementation of the Gosu interface `NameOwner` to get values for some or all of the following field properties:

```
editable
label
required
value
available
```

Gosu name formatter

PolicyCenter uses the Gosu class `NameFormatter` to format read-only name display fields. You can extend `NameFormatter` to handle additional regions. See "NameFormatter class" on page 83.

Name owners

NameOwner is the interface for a helper object that is passed to the modal PCF files GlobalPersonNameInputSet and GlobalContactNameInputSet. The helper object provides a way to set and get a single name on the enclosing entity. Typically you extend NameOwnerBase, which implements NameOwner.

While NameOwner provides methods for setting a field to be required or visible, generally it is better not to modify or override methods such as isVisible in NameOwnerBase. For most purposes, specifying the set, required, or hidden fields is more easily controlled by overriding property getters such as RequiredFields and HiddenFields declared in NameOwner. You override these property getters with values such as NameOwnerFieldId.REQUIRED_NAME_FIELDS or NameOwnerFieldId.NO_FIELDS.

Following are some properties on NameOwner:

Property	Description
ContactName	Sets (or retrieves) a single contact name on the enclosing entity, Contact. For example, the Name field of GlobalContactNameInputSet.default has the value nameOwner.ContactName.Name
PersonName	Sets (or retrieves) a single person name on the enclosing entity, Person. For example, the First Name field of GlobalPersonNameInputSet.default has the value nameOwner.PersonName.FirstName
HiddenFields	Set of name fields that ClaimCenter hides (does not show) in the application interface.
RequiredFields	Set of name fields for which the user must supply a value.

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

- NameOwnerBase
- AffinityGroupNameOwner
- BasicNameOwner
- ContactNameNoSummaryOwner
- ContactNameOwner
- GroupNameOwner
- MVRDriverNameOwner
- NewAffinityGroupNameOwner
- OrganizationNameOwner
- PLPersonNameSearchOwner
- RequiredBasicNameOwner
- UserSearchNameOwner

In the base configuration, ContactManager provides the following classes that implement NameOwner or extend a class that implements NameOwner:

- NameOwnerBase
- ABUserNameOwner
- ContactNameOwner
- PLPersonNameSearchOwner
- SearchNameOwner
- UserSearchNameOwner

NameFormatter class

PolicyCenter displays read-only name information in various screens. It uses class gw.name.NameFormatter to manage and format these read-only strings.

The NameFormatter class is used to convert Contact names to strings for globalization. If you change, add, or delete name columns of the Contact entity or its subentities, you must also update this class.

Note: ContactManager uses NameFormatter to convert names used in ABContact and subentities of ABContact to strings for globalization.

Additionally, if you add a new region definition, you might need to update the internalFormat method of this class, which uses fields defined in the NameOwnerFieldId class.

The class contains several different versions of the format method with different signatures, for example:

- The following version of the format method formats a name as text and includes all fields that are used for the current region. This case is the common one. This method has the following signature:

```
public function format(name : ContactNameFields, delimiter : String) : String {
  _filter = \ fieldId : NameOwnerFieldId -> { return true }
  return internalFormat(name, delimiter)
}
```

- The following version of the format method formats a name as text and includes only the specified set of fields from the current region. This method has the following signature:

```
public function format(name : ContactNameFields,
      delimiter : String, fields : Set<NameOwnerFieldId>) : String {
  _filter = \ fieldId : NameOwnerFieldId -> { return fields.contains(fieldId) }
  return internalFormat(name, delimiter)
}
```

The parameters for these methods have the following meanings:

Parameter	Description
name	The Contact name to format.
delimiter	The delimiter that separates elements of the name, typically " ".
fields	The set of Contact fields to include in the name.

PolicyCenter modal PCF file

The following PolicyCenter modal PCF file uses NameFormatter:

- GlobalPersonNameInputSet sets the following Value for the Name field:

```
new gw.api.name.NameFormatter().format(nameOwner.PersonName, " ")
```

The following PolicyCenter Gosu class and Gosu enhancements call NameFormatter:

```
gw.contact.AbstractContactResult.gs
gw.plugin.contact.impl.ContactEnhancement.gsx
gw.plugin.contact.impl.ContactKanjiEnhancement.gsx
```

The following PolicyCenter display names call NameFormatter to return Contact names:

- displaynames/CommercialDriver.en returns:

```
new NameFormatter().format(
        person, " ",
        NameOwnerFieldId.DISPLAY_NAME_FIELDS)
```

- displaynames/Company.en returns:

```
new NameFormatter().format(contact, " ")
```

- displaynames/Contact.en returns one of the following, based on Contact subtype:

```
new NameFormatter().format(
       person, " ",
       NameOwnerFieldId.DISPLAY_NAME_FIELDS)
new NameFormatter().format(contact, " ")
```

- displaynames/Place.en returns:

```
new NameFormatter().format(contact, " ")
```

- `displaynames/PolicyContactRole.en` returns one of the following, based on `Contact` subtype:

```
new NameFormatter().format(
       person, " ",
       NameOwnerFieldId.DISPLAY_NAME_FIELDS)
new NameFormatter().format(contact, " ")
```

ContactManager modal PCF file

The following ContactManager modal PCF file uses `NameFormatter`:

- `GlobalPersonNameInputSet` sets the following `Value` for the `Name` field:

```
new gw.api.name.NameFormatter().format(
       nameOwner.PersonName, " ")
```

The following ContactManager display names call `NameFormatter` to return `ABContact` names:

- `displaynames/ABCompany.en` returns:

```
new NameFormatter().format(contact, " ")
```

- `displaynames/ABContact.en` returns one of the following, based on `ABContact` subtype:

```
new NameFormatter().format(
       person, " ",
       NameOwnerFieldId.DISPLAY_NAME_FIELDS)
new NameFormatter().format(contact, " ")
```

- `displaynames/ABPlace.en` returns:

```
new NameFormatter().format(contact, " ")
```

See also

- “NameOwnerFieldId class” on page 85
- “Setting up additional region and name configurations” on page 80

NameOwnerFieldId class

Guidewire provides a `gw.api.name.NameOwnerFieldId` class that provides type safety for `Name` entity fields. If you extend the `Contact` entity or the `ABContact` entity with a new name column, you must add the new column to this class as a new constant.

Constants that represent name fields

In the base configuration, `NameOwnerFieldId` provides the following constants that represent name fields.

Available constants for person name

```
PREFIX
FIRSTNAME
MIDDLENAME
PARTICLE
LASTNAME
SUFFIX
FIRSTNAMEKANJI
LASTNAMEKANJI
```

Available constants for non-person name

```
NP_NAME
NAMEKANJI
```

For example, you add the new `Contact` name column `MyNameColumn` to the `NameOwnerFieldId` class with the following code:

```
public static final var MYNAMECOLUMN :
    NameOwnerFieldId = new NameOwnerFieldId("MyNameColumn")
```

Constants that use name field ID constants

Class `NameOwnerFieldId` also defines a set of constants that use the following name field ID constants:

```
public final static var ALL_PCF_FIELDS : Set<NameOwnerFieldId> =
  { PREFIX, FIRSTNAME, MIDDLENAME, PARTICLE, LASTNAME, SUFFIX, FIRSTNAMEKANJI,
    LASTNAMEKANJI, NP_NAME, NAMEKANJI }.freeze()

 /** Fields used for non-Person names */
public final static var ALL_CONTACT_PCF_FIELDS : Set<NameOwnerFieldId> =
  { NP_NAME, NAMEKANJI }.freeze()

 /** Required fields (union of fields for both Persons and non-Persons) */
public final static var REQUIRED_NAME_FIELDS : Set<NameOwnerFieldId> =
  { LASTNAME, NP_NAME }.freeze()

 /** Fields shown in display names */
public static final var DISPLAY_NAME_FIELDS : Set<NameOwnerFieldId> = {
  FIRSTNAME, PARTICLE, LASTNAME, SUFFIX }.freeze()

 /** Fields for simple name */
public final static var FIRST_LAST_FIELDS : Set<NameOwnerFieldId> =
  { FIRSTNAME, LASTNAME }.freeze()

 public final static var HIDDEN_FOR_SEARCH : Set<NameOwnerFieldId> =
  { PREFIX, MIDDLENAME, PARTICLE, SUFFIX }.freeze()
```

You can add a new `Contact` name column to one of these constants by using its constant name, such as `MYNAMECOLUMN`. For example:

```
public final static var ALL_PCF_FIELDS : Set<NameOwnerFieldId> =
  {PREFIX,
   FIRSTNAME,
   MIDDLENAME,
   MYNAMECOLUMN,
   PARTICLE,
   LASTNAME,
   SUFFIX,
   FIRSTNAMEKANJI,
   LASTNAMEKANJI,
   NP_NAME,
   NAMEKANJI
  }.freeze()
```

chapter 14

Working with Kanji fields

Kanji fields are available in various entities, such as `Contact` and its subtypes and `Address`. Additionally, Kanji fields are supported by contact search entities and classes.

You can enable indexes for these fields to improve search performance.

- For information on Kanji fields used in names, see "Configuring name information" on page 77.
- For information on Kanji fields used in addresses, see "Configuring address information" on page 105.

Enabling indexes for Kanji fields

Indexes are used to improve search performance and, where needed, to provide uniqueness. For Kanji fields, there are a number of indexes provided as commented-out code in the base configuration that must be enabled if your installation uses Kanji fields. If your installation does not use the Kanji fields provided in the base configuration, do not enable these indexes.

You must enable Kanji indexes in both PolicyCenter and ContactManager.

Enable Kanji indexes in PolicyCenter

About this task

You will need to restart PolicyCenter and ContactManager after making these changes. While you cannot make these changes in Guidewire Studio, you can use Studio to find all the files with commented-out Kanji indexes in them. In Studio, press `Ctrl+Shift+F`, and then search for the string `KanjiIndexDefinition` in the entire project. You can then open each file in a text editor and make the changes.

Procedure

1. In your file system, navigate to the files listed in step 3 and open them in an editor, not in Guidewire Studio.
2. Search for a comment that starts with `KanjiIndexDefinition`.
3. Uncomment the following Kanji index definitions:

   `/modules/configuration/config/extensions/entity/Contact.Global.etx`

   Uncomment the index named `CompanyNameK1`.

   `/modules/configuration/config/extensions/entity/Group.Global.etx`

   Uncomment the index named `group1K`.

Enable Kanji indexes in ContactManager

About this task

You will need to restart ContactManager after making these changes. While you cannot make these changes in Guidewire Studio™ for ContactManager, you can use Studio to find all the files with commented-out Kanji indexes in them. In Studio, press `Ctrl+Shift+F`, and then search for the term `KanjiIndexDefinition` in the whole project.

Procedure

1. In your file system, navigate to the files listed in step 3 and open them in an editor, not in Guidewire Studio.
2. Search for a comment that starts with `KanjiIndexDefinition`.
3. Uncomment the following Kanji index definitions:

   `ContactManager/modules/configuration/config/extensions/entity/Contact.Global.etx`

   Uncomment the index named `CompanyNameK1`

   `ContactManager/modules/configuration/config/extensions/entity/Group.Global.etx`

   Uncomment the index named `group1K`.

   Comment out the index named `group1`.

   `ContactManager/modules/configuration/config/extensions/entity/ABPerson.etx`

   Uncomment the following indexes:

   - `ABPersonKWFNCSK`
   - `ABPersonKWFNPCK`
   - `CFindDupeNameK`

   `ContactManager/modules/configuration/config/extensions/entity/Address.etx`

   Uncomment the indexes named `address2K` and `address6K`.

chapter 15

Working with the Japanese Imperial Calendar

PolicyCenter provides a Japanese Imperial calendar date picker that you can display in the user interface. You enable this feature through configuration. After you enable it, you can specify the default calendar type for an entity property.

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

- Reiwa
- Heisei
- Showa
- Taisho
- Meiji

The Japanese Imperial Calendar has the following usage limitation:

- Copy and paste of a date string into the date input field can often produce an incorrect entry.
- Your web browser must be using the same time zone as the PolicyCenter server. If the time zone for the web browser is different from the server, the calendar date picker does not save the correct date.

Configuring Japanese dates

You configure PolicyCenter display of Japanese Imperial calendar dates by specifying the `<JapaneseImperialDateFormat>` element in `localization_`*`localeCode`*`.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 `short="G yy/MM/dd"`.
`yearSymbol`	PolicyCenter uses the Japanese `yearSymbol` as a signal to render the Japanese Imperial calendar date picker.

To see the default setting for the Japanese Imperial calendar in the PolicyCenter base configuration, open file `localization_ja_JP.xml` in following Guidewire Studio location:

configuration→config→Localizations

Set the Japanese Imperial Calendar as the default for a region

About this task

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 `enableJapaneseCalendar` of the `<GWLocale>` element to `true`. These attributes determine whether or not to enable the Japanese calendar for this region.

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.

Procedure

1. In the Guidewire Studio **Project** window, navigate to **configuration→config→Localizations** and double click `localization_ja_JP.xml` to open it in the editor.
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">
```

Result

After setting these calendar attributes, if the user chooses this region in PolicyCenter, all dates are shown as Japanese Imperial Calendar dates.

See also

- “Working with regional formats” on page 67

Set fields to display the Japanese Imperial Calendar

About this task

You can set a field of an entity to display the Japanese Imperial calendar when it is used in the PolicyCenter user interface. For example, you want to add an additional field to the `Activity` object that uses the `japaneseimperialdate` data type.

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

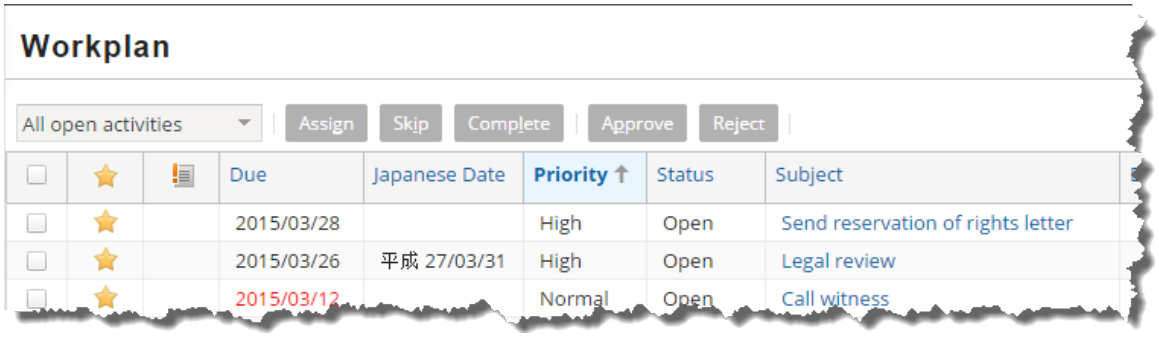
Procedure

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

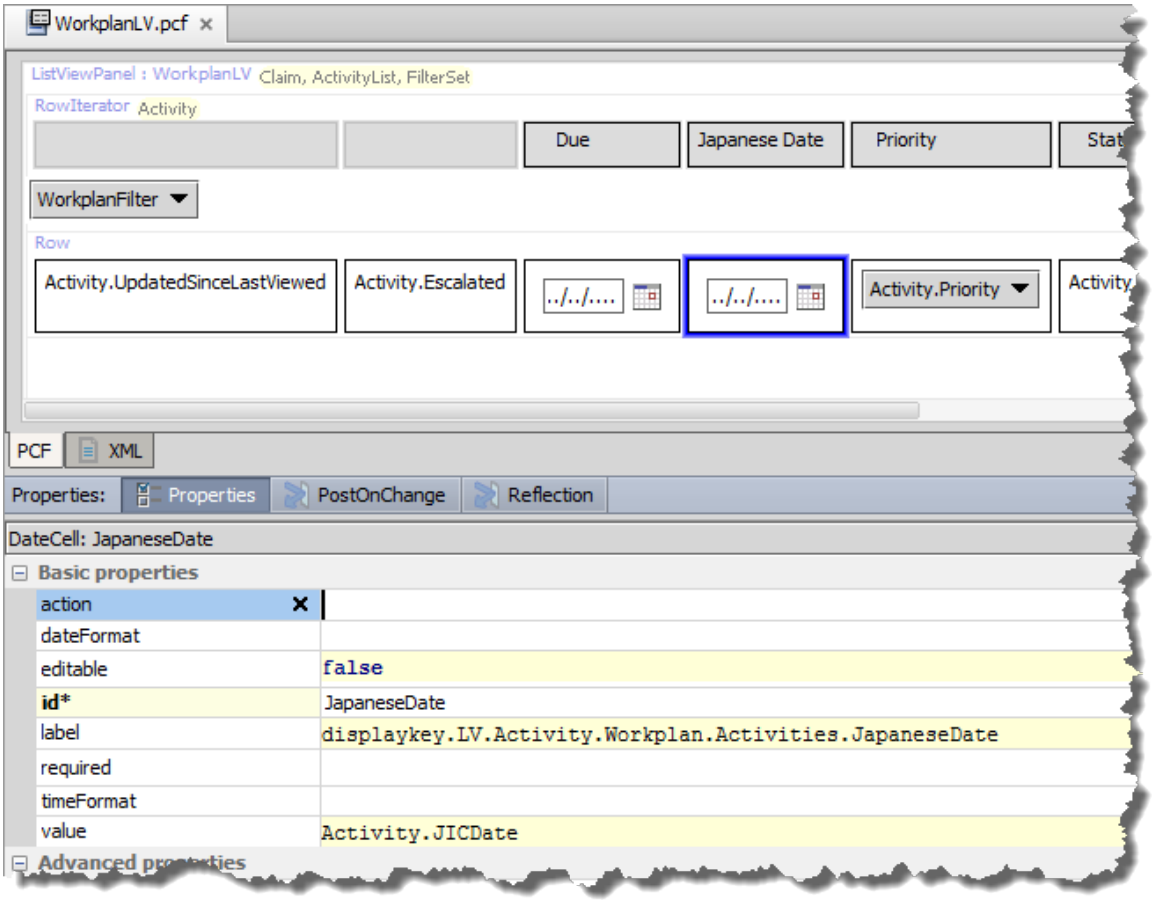
   To be useful, you need to use this field to display a value in the Japanese Imperial calendar format.

   For example, in the **Workplan** screen for a claim, you can add a **Japanese Date** field.

   In the following screen you see a Gregorian date for the **Due** column and a Japanese Imperial date for the **Japanese Date** column. This result is possible when you do not define the Japanese Imperial Calendar as the default for the region. The user must still set the regional format to Japan, however. If you had defined the Japanese Imperial Calendar as the default calendar in `localization_ja_JP.xml`, all dates would be in Japanese Imperial format. “Set the Japanese Imperial Calendar as the default for a region” on page 90.



In the `WorkplanLV` PCF file, the field definition looks similar to the following:



Set keyboard shortcut for Imperial Era

In working with the Japanese Imperial Calendar inputs, you can use a keyboard shortcut to select the Imperial Era.

About this task

Entering the first character of the English translation of the Japanese Imperial Calendar (JIC) era selects that era in the JIC date input field. For example, entering H as soon as the date input receives focus automatically selects the Heisei Era. In a similar fashion, entering R in the data input field automatically selects the Reiwa Era.

You must configure this functionality by adding a `<JapaneseImperialDateFormat>` element to localization file `localization_en_US.xml`.

Procedure

1. In the Guidewire Studio **Project** window, navigate to the following location and open file `localization_en_US.xml`:

   configuration→config→Localization

2. Inside the `<GWLocale>` element, starting on a new line immediately after the last item in the group, add a `<JapaneseImperialDateFormat>` element, for example:

```
<GWLocale code="en_US" name="United States (English)" typecode="en_US">
        <DateFormat long="E, MMM d, yyyy" medium="MMM d, yyyy" short="MM/dd/yyyy"/>
        <TimeFormat long="h:mm:ss a z" medium="hh:mm:ss a" short="h:mm a"/>
        <NumberFormat decimalSymbol="." thousandsSymbol=","/>
        <CurrencyFormat negativePattern="($#)" positivePattern="$#" zeroValue="-"/>
        <JapaneseImperialDateFormat
          long=""
          medium=""
```

```
          short=""
        />
    </GWLocale>
```

See localization file `localization_ja_JP.xml` for an example of what to enter for the element attributes. These three element attributes are required.

3. Save your work.

Next steps

You must stop and restart the application server for this change to take effect.

chapter 16

Configuring geographic data

You can configure the typelists `Jurisdiction`, `Country`, and `State`, and configuration file `zone-config.xml` for the territorial data in your instance of PolicyCenter.

About 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:

https://www.iso.org/obp/ui/

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 address formats by country

The `country.xml` files in the country folders define settings that control the appearance of address information for each country in PolicyCenter. To access the `country.xml` file for a specific country, navigate to the following location in the Project window in Guidewire Studio for PolicyCenter:

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

Setting <Country> attributes

You can set the following attributes on the `<Country>` element.

Attribute	Description
PCFMode	PCF mode for GlobalAddressInputSet.pcf. In the base configuration, Guidewire provides three modes for showing address information. The base configuration modes are: • BigToSmall – Used to format addresses for Japan • PostCodeBeforeCity – Used to format addresses for France and Germany • default – Used to format all other addresses The default value is default. You can add new PCF modes and use them in the file country.xml.
postalCodeDisplayKey	Name of the display key to use for the postal code label in PolicyCenter. For example: • In the United States (US) – Web.AddressBook.AddressInputSet.ZIP • In Japan (JP) – Web.AddressBook.AddressInputSet.Postcode The default value is Web.AddressBook.AddressInputSet.PostalCode.
cityDisplayKey	Name of the display key to use for the city label for a country. For example: • In Great Britain (GB) – Web.AddressBook.AddressInputSet.TownCity The default value is Web.AddressBook.AddressInputSet.City.
stateDisplayKey	Name of the display key to use for the label designating the major administrative subdivisions in a country. For example: • In the United States (US) – Web.AddressBook.AddressInputSet.State • In Canada (CA) – Web.AddressBook.AddressInputSet.Province • In Japan (JP) – Web.AddressBook.AddressInputSet.Prefecture The default value is Web.AddressBook.AddressInputSet.State.
visibleFields	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: • Country • AddressLine1 • AddressLine2 • AddressLine3 • City • State • PostalCode Any address field that you designate as visible in this file must correspond to the constants defined in AddressOwnerFieldId.gs.

Display key values

For attributes that take display key values, PolicyCenter uses display properties files defined in **Resource Bundle 'display'** in the **Localizations** folder. For example:

- PolicyCenter uses `display.properties` for U.S. English
- PolicyCenter uses the appropriate `display_`*`LanguageCode`*`.properties` file, such as `display_ja_JP.properties` for Japan.

If there is no existing `display_`*`LanguageCode`*`.properties` file for a region, PolicyCenter uses the default `display.properties` file.

See also

- “Configuring address data and field order for a country” on page 108
- “Address modes in page configuration” on page 111
- “Localizing display keys” on page 35.

Setting the default application country

You set the default application country using configuration parameter `DefaultCountryCode`, stored in the file `config.xml`. This parameter determines which country PolicyCenter uses if the user does not specify a country for

an address. PolicyCenter also uses the value of this parameter as the default country code for any new addresses that it creates. The country code must be a valid ISO country code that exists as a typecode in the `Country` typelist.

See the following page to search a list of ISO country codes:

https://www.iso.org/obp/ui

See also

- *Configuration Guide*.

Configuring jurisdiction information

Guidewire applications divide jurisdictions into several areas:

- **National jurisdictions** – Japan or France, for example
- **State or province jurisdictions** – Idaho, U.S.A, or, Ontario, 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 state information

State information includes both the name of the state and the usual abbreviation of the state name.

The `State` typelist

PolicyCenter provides a `State` typelist to represent states, prefectures, or provinces. The typecodes in this typelist have `Name` values that are full, unabbreviated names, such as Washington for the US typecode `WA` and Western Australia for the AU typecode `AU_WA`.

The `StateAbbreviation` typelist

Abbreviations corresponding to `State` typecodes are defined in the typelist `StateAbbreviation`. This typelist provides a set of typecodes with `Name` values that are abbreviations for states or provinces.

Note: Japan (JP) does not use abbreviations for its prefecture names.

Mapping between state names and state abbreviations

For countries that use state abbreviations, such as AU and US, there is a one-to-one relationship between the country's typecodes for `State` and `StateAbbreviation`. This relationship supports distinguishing between abbreviations that have the same value in more than one country. For example, the abbreviation WA is used both in the United States for the state of Washington and in Australia for the state of Western Australia.

For the countries defined in the `StateAbbreviation` typelist, you can use methods on the `State` and `StateAbbreviation` typelists to get abbreviations and state names.

State typelist abbreviation methods

The following methods used by the `State` typelist are defined in `gw.entity.GWStateEnhancement.gsx`.

State.getAbbreviation

- **Parameters** – none
- **Return value** – `typekey.StateAbbreviation`
- **Comments** – Returns the abbreviation typekey for the state as defined in the `StateAbbreviation` typelist. This value can vary based on the current language setting. Returns `null` if no abbreviation is defined, such as for Japanese prefectures.

State.getState

- **Parameters** – `country:Country, anAbbreviation:String`
- **Return value** – `typekey.State`
- **Comments** – Look up a `State` by using the country typecode and the country-dependent, localized state abbreviation. For example, if the language is U.S. English:
  - `getState(TC_AU, "WA")` returns Western Australia.
  - `getState(TC_US, "WA")` returns Washington.

State.getStateAbbreviation

- **Parameters** – `country:Country, anAbbreviation:String`
- **Return value** – typekey.`StateAbbreviation`
- **Comments** – Looks up a `StateAbbreviation` by using the country and the country-dependent, localized state abbreviation. For example, if the language is U.S. English:
  - `getStateAbbreviation(TC_AU, "WA")` returns the `StateAbbreviation` typekey for Western Australia.
  - `getStateAbbreviation(TC_US, "WA")` returns the `StateAbbreviation` typekey for Washington.

StateAbbreviation typelist abbreviation method

There is one method used by the `StateAbbreviation` typelist that is defined in `gw.entity.GWStateAbbreviationEnhancement.gsx`:

StateAbbreviation.getState

- **Parameters** – none
- **Return value** – `typekey.State`
- **Comments** – Returns the `State` typekey associated with the current `StateAbbreviation` typecode.

Zone configuration

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

Overview of configuring zones

To use zones in your application, you must import zone files, either files that you have created or files that you have purchased from a vendor. For example, your company might have bought a dataset from a company like GreatData. You then use this dataset to plan your zone configuration. For more information, see “Importing address zone data” on page 103

Note: PolicyCenter ships with sample address data for Australia, Canada, Canada with FSA, Germany, France, Japan, and the United States of America. This data is for use in testing and is not suitable for a production environment.

In planning, consider what parts of an address you need to auto-fill and the format of the ZIP or postal codes. Also, note if any of the values are not unique for a country. For example, the country has two cities with the same name.

While most countries have a similar format for addressing, there can be differences. For example, France uses a five-digit postal code plus a city. Additionally, France has a small locality format that includes more information.

You can define multiple zone types for a country. For example, you can divide your country into zones by county, state, and city. Most cities are in a single zone, such as a state in the United States or a province in Canada.

Zone types are defined as typecodes in the `ZoneType` typelist. You use these typecodes as elements in the `zone-config.xml` file. You can add your own zone types to this typelist.

A zone type can be defined to refer to another zone types by using a link. PolicyCenter uses links to look up a zone based on another zone. An example is the relationship between the ZIP code and the state for US zones. Your configuration could link ZIP code to state. For example, given the ZIP code of 94404, the application could auto-fill the corresponding state, California.

You do not necessarily need to define all the `Zone` subelements for a zone. For example, you can define a zone as the entire country, such as the following definition for all of Australia:

```
<Zones countryCode="AU"/>
```

Location of zone configuration files

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

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` – Unites States of America

In each `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 a zone from address data
- How PolicyCenter imports data from zone data files

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>
```

See also

- For information on the zone import command, see the *System Administration Guide*.

<Zones> element

This element defines the largest region, a country. The `zone-config.xml` file 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`	This required element defines the country to which this zone configuration applies.
`dataFile`	For Guidewire internal use only. Do not set this attribute.

<Zone> element

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	This optional attribute 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.
	A <Zone> 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 <Zones> element:
	`<Zone code="fsa" regionMatchOrder="1" granularity="1">`
	`  <ZoneCode> postalcode.substring(0,3) </ZoneCode>`
	`  <AddressZoneValue> Address.PostalCode.substring(0,3) </AddressZoneValue>`
	`</Zone>`
	Both the <ZoneCode> and <AddressZoneValue> elements extract data from the actual address data by parsing the data into substrings.
	Notes:
	• Specify at least one <Zone> 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.
	• Import address zone data on first installing Guidewire PolicyCenter, and then at infrequent intervals thereafter as you update zone definitions.

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.

regionMatchOrder	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 <Zone> attributes for a county:
	`<Zone`
	`  code="county"`
	`  fileColumn="4"`
	`  regionMatchOrder="2"`
	`  granularity="3" >`
	`  ...`
	`</Zone>`
	Setting the regionMatchOrder to 2 means that PolicyCenter matches county data second, after another zone, while matching a user to a location.
	The county value also:
	• Appears in the fourth column of the data file.
	• Is third in granularity, one size less than a state—granularity 4—and one size more than a city—granularity 2.

unique	This optional attribute specifies whether the 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.
	To differentiate zones that can be identical—not unique—use a <ZoneCode> element to construct a zone expression that uniquely identifies that zone data. For example, you can combine the county name with the state name to make a unique identifier by defining the following <ZoneCode> subelement of <Zone>:
	`<ZoneCode>`
	`  state + ":" + county`
	`</ZoneCode>`
	See also, “<ZoneCode> element” on page 102.

<ZoneCode> element

The <ZoneCode> element is a subelement of <Zone> used to differentiate identical zone information. It is possible that zone information is not 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:

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

This expression specifies that the State value must be concatenated 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> element" on page 100

<Links> element

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.

Note: An address-config.xml file can define auto-fill elements that use these links. See "Configuring autofill and autocompletion in address-config.xml" on page 113.

The <Link> element has the following attributes:

Attribute	Description
lookupOrder	This optional attribute specifies 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 lookupOrder value, PolicyCenter uses the order that appears in the file.
toZone	This required attribute defines a relationship to another zone for PolicyCenter to use if the value of the <Zone> attribute code 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> element" on page 100

<AddressZoneValue> element

The `<AddressZoneValue>` element is a subelement of `<Zone>`. This 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> element" on page 100

Importing address zone data

You can import address zone data when you first install the product and at infrequent intervals thereafter as you receive data updates. You use the `zone_import` command to import zone data into staging tables, and then you use the `table_import` command to import the staging table data.

In the base configuration, Guidewire provides the following sample zone data files in **configuration→config→geodata**:

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

Guidewire provides these files for testing purposes to support auto-fill and auto-complete when users enter addresses. You must import one of these files to use it in testing those features. This data is provided on an as-is basis. The provided zone data files are not complete and might not include recent changes.

The formatting of individual data items in these files might not conform to your internal standards or the standards of third-party vendors that you use. For example, the names of streets and cities are formatted with mixed case letters, but your standards might require all upper case letters.

The `US-Locations.txt` file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings. You can edit this file to conform to your particular address standards, and then import that version of the file.

Guidewire does not provide complete and up-to-date zone data files that you can import with the `zone_import` command. You must create these files or obtain them from vendors, like GreatData.

See also

- For information on the zone import command, see the *System Administration Guide*

Basic zone types

You can define basic zone types for your country configuration in the `ZoneType` typelist.

ZoneType typelist

The base configuration of the `ZoneType` typelist provides typecodes for Australia, Canada, France, Germany, Japan, and the United States. The base PolicyCenter configuration provides zones for the following items:

- Country
- City
- City Kanji
- County
- State
- Prefecture
- Province
- Postal code
- ZIP code
- FSA
- Post code area
- Post code region

FSA stands for Forward Sortation Area, the first three letters of a Canadian postal code.

Add a new zone type

Procedure

1. In the Studio **Project** window, navigate to the following location:

   configuration→config→Extensions→Typelist
2. Double-click `ZoneType.ttx` to open it for editing.
3. Edit the typelist and enter the necessary information for a new zone type code.

chapter 17

Configuring address information

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

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

- "Addresses and the AddressFormatter class" on page 106
- "Address modes in page configuration" on page 111
- "Address owners" on page 111
- "AddressOwnerFieldId class" on page 112

Overview of 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

Overview of 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 folders under the `geodata` folder. Each file is 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 also

- "Configuring address data and field order for a country" on page 108.

Overview of 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 Australia, Canada, Great Britain, and the United States.
- `GlobalAddressInputSet.BigToSmall` – Used for Japan.
- `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 109.

See also

- For more information on `GlobalAddressInputSet`, see "Address modes in page configuration" on page 111.
- For more information on modal PCF files, see the *Configuration Guide*.

Addresses and the AddressFormatter class

The `AddressFormatter` class is used to convert addresses to localized strings for display as read-only address information. 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 by using the `gw.address.AddressFormatter` class.

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 method parameters have the following meanings:

Parameter	Description
`address`	The address to format.
`delimiter`	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.
`fields`	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 because it requires no additional parameters.
- `Address.addressString` is the more general form of usage with an address, especially 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 110

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 that you can access in Guidewire Studio™ 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→*CountryCode*	`address-config.xml` – Defines formats to use for address autofill 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 "Zone configuration" on page 98	
configuration→config→geodata	*CountryCode*`-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 110.

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, PolicyCenter stores the `country.xml` file for Japan in the following folder:

configuration→config→geodata→JP

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

Attribute	Description	More information
`visibleFields`	Which address fields to display	“Visible fields attribute of the country XML file” on page 109
`PCFMode`	Order in which to display address fields	“PCFMode attribute of the country XML file” on page 109
`postalCodeDisplayKey`	Label for the postal code field	“Postal code display key attribute of the country XML file” on page 109
`stateDisplayKey`	Label for state or province field	“State display key attribute of the country XML file” on page 110

Visible fields attribute of the country XML file

Attribute `visibleFields` on the `<country>` element in file `country.xml` defines the set of address fields that are visible for this country. For example:

France

```
visibleFields="Country,AddressLine1,AddressLine2,AddressLine3,PostalCode,City,CEDEX,CEDEXBur
eau"
```

Japan

```
visibleFields="Country,PostalCode,State,City,CityKanji,AddressLine1,AddressLine1Kanji,Addres
sLine2,AddressLine2Kanji"
```

United States

```
visibleFields="Country,AddressLine1,AddressLine2,AddressLine3,City,State,PostalCode,County"
```

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

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

See also

- “AddressOwnerFieldId class” on page 112

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

```
PCFMode="PostCodeBeforeCity"
```

Japan

```
PCFMode="BigToSmall"
```

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 111

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

```
postalCodeDisplayKey="Web.AddressBook.AddressInputSet.Postcode"
```

United States

```
postalCodeDisplayKey="Web.AddressBook.AddressInputSet.ZIP"
```

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

```
stateDisplayKey="Web.AddressBook.AddressInputSet.Prefecture"
```

United States

```
stateDisplayKey="Web.AddressBook.AddressInputSet.State"
```

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

Additional country and address configurations

Before you begin

If you add a new country, you must do configuration in addition to the configurations required for the `country.xml` file, described in "Configuring the Country XML file" on page 108.

Procedure

1. You must 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.

2. 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:

   `gw.address.AddressFormatter`

   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.

   `Address.en`

   Modify the definition of the display name through the Entity Names editor. See the *Configuration Guide* for details.

   `AddressOwnerFieldID.gs`

   Add a variable for the new `Address` column to this class.

   See "AddressOwnerFieldId class" on page 112.

   `gw.policylocation.PolicyLocationEnhancement`

   Do the following:

   a. Modify the `PolicyLocation.addressString` method as needed and populate any added column before calling the `AddressFormatter` class.

   b. Add the `get` and `set` values for the new internal address columns.

   `gw.policyaddress.PolicyAddressEnhancement`

   Do the following:

   a. Modify the `PolicyAddress.addressString` method as needed and populate any added column before calling the `AddressFormatter` class.

   b. Add the `get` and `set` values for the new internal address columns.

c. Add the new columns to the `copyToNewAddress` and `copyFromAddress` functions.

See also

- "Addresses and the AddressFormatter class" on page 106
-

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
`GlobalAddressInputSet.BigToSmall`	• Japan
`GlobalAddressInputSet.PostCodeBeforeCity`	• France • Germany
`GlobalAddressInputSet.default`	• Australia • Canada • Great Britain • United States

To see the `GlobalAddressInputSet` PCF files, open Guidewire Studio™ and navigate in the **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. Following are some properties on `AddressOwner`.

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

Address autocompletion and autofill

PolicyCenter supports automatic fill-in and completion of address information.

Address autofill

The *autofill* feature enables you to enter a value in one address field in a Guidewire application and have the application fill in other address fields automatically.

For example, if configured, entering a postal code causes PolicyCenter to fill in the city and state or province fields automatically. For example, the user enters a postal code of 99501 for a United States address. PolicyCenter sets the city to Anchorage, the state to Alaska, and the county to Anchorage.

To trigger autofill, the user must enter a value and then navigate away from the initial address field, such as the postal code field. It is also possible to trigger this functionality by using the autofill icon next to certain address fields. There must also be a unique match, such as state and city to postal code.

Address autocompletion

The *autocomplete* feature enables you to enter the first few characters of a field and see a drop-down list showing the matching values. 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 names of cities in California that start with `Pa`, such as:

- Pacifica
- Pacoima
- Palm Springs
- Palo Alto
- Panorama City
- Pasadena
- Paso Robles

The user can then select one of the items on the drop-down list to populate the address field.

Configuring autofill and autocompletion in address-config.xml

To configure the fields that support address autofill and autocompletion for a country, configure that country's `address-config.xml` file.

> **Note:** The `address-config.xml` file uses zones defined in `zone-config.xml`. See "Zone configuration files" on page 99.

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

configuration→config→geodata→JP

The following default configuration for the United States, in **configuration→config→geodata→US**, shows elements that you can use in this file:

```
<AddressDef name="US">
  <Match>
    <Field name="Country" value="US"/>
  </Match>
  <Fields>
    <Field name="City" zonecode="city">
      <AutoFillFromZone code="zip"/>
      <AutoFillFromZone code="state"/>
    </Field>
    <Field name="County" zonecode="county">
      <AutoFillFromZone code="zip"/>
      <AutoFillFromZone code="city"/>
      <AutoFillFromZone code="state"/>
    </Field>
    <Field name="State" zonecode="state">
      <AutoFillFromZone code="zip"/>
      <AutoFillFromZone code="city"/>
    </Field>
    <Field name="PostalCode" zonecode="zip">
```

```
      <AutoFillFromZone code="city"/>
      <AutoFillFromZone code="state"/>
      <ValidatorDef
         value="[0-9]{5}(-[0-9]{4})?"
         description="Validator.PostalCode.US"
         input-mask="#####-####"/>
    </Field>
  </Fields>
</AddressDef>
```

The following table describes the elements in the `address-config.xml` file.

`AddressDef`	The name of the address format. This element takes a `name` attribute and an optional `priority` attribute. By convention, the `name` is the country code. A country can have more than one address format—for example, it is possible that different regions have different formats. The `priority` attribute specifies which address definition has priority if several of them match. The highest priority is 1, the next lower one is 2, and so on.
`Match`	Each `AddressDef` must contain one `Match` element. PolicyCenter matches only on the country. The `Match` element contains a `Field` subelement that defines the `name` and `value` attributes that the application uses to determine which definition applies.
`Fields`	Each `AddressDef` contains a single `Fields` element that contains a list of the address `Field` elements.
`Field`	Specifies an address field. Each field takes a name that must match a property on the `Address` entity. The `Field` element can appear in the `Match` element or the `Fields` element. In the `Match` element, the `Field` element has a required `name` attribute and an optional `value` attribute, which is the `code` value from the `Country` typelist. It has no child elements. In the `Fields` element, the `Field` element has a required `name` attribute and optional `zonecode` and `autoCompleteTriggerChars` attributes. It can also have the child elements `AutoFillFromZone` and `ValidatorDef`. • The `value` attribute is a valid value for the `Field`. This value is usually a code value from a typelist, such as a state typecode from the typelist for a `Field` with the `name` set to `State`. • The `zonecode` attribute corresponds to a `Zone` code configured for the given country in `zone-config.xml`. This value links the `Address` configuration to the `Zone` configuration. For information on zone configuration, see “Zone configuration files” on page 99. • The `autoCompleteTriggerChars` attribute specifies the number of characters to enter before the application triggers autocomplete. The default value is 0 (zero).
`AutoFillFromZone`	Specifies a field that is examined for autofill and autocompletion of the field. PolicyCenter uses the zone `code` to look up the field `value`. See “Zone configuration files” on page 99.
`ValidatorDef`	Contains information for validating the field in the optional attributes `value`, `description`, and `input-mask`. **Note:** Do not define field validators in `address-config.xml`. Guidewire recommends that you define them in `fieldvalidators.xml`. See the *Configuration Guide*.

The previous `address-config.xml` example defines a `Match` on the country with the match value as `US`. Additionally, it defines four fields:

- `City`
- `County`
- `State`
- `PostalCode`

Each field defines one or more `AutoFillFromZone` elements. Looking at the `County`, you can see that the application fills the `County` from the `zip`, the `city`, and the `state`. Each `AutoFillFromZone` entry must correspond to a `<Link>` definition in `zone-config.xml`, as described in “Zone configuration files” on page 99. That file specifies that autofill can use ZIP code and city for the look-up operation.

As PolicyCenter loads `address-config.xml`, it validates the configuration. PolicyCenter verifies that every `Field` element, regardless of whether it is defined in `Match` or `Fields`, corresponds to a field on the `Address` entity. Then, PolicyCenter verifies that each `AutoFillFromZone` element references a zone in `zone-config.xml`.

See also

- *Integration Guide*

Configuring autofill and autocompletion in a PCF file

You can configure autofill and autocomplete after you have configured zone mapping and addressing and have imported your zone data. PolicyCenter uses the following main classes for address autofill and autocompletion.

`gw.api.contact.AddressAutocompleteHandler`

Class provides methods for getting matching values from the database.

`gw.api.contact.AddressAutocompleteUtil`

Class supplies methods for automatically completing an address field.

You can integrate autofill and autocompletion into a PCF file in the following ways.

Complete a field automatically after you type in a few characters

For example, if you enter `941` for the ZIP code, the autocomplete list shows `94110`, `94111`, `94112`, and so forth.

Fill other address fields after you complete a field and tab out of it

For example, after you enter the ZIP code `94115`, autofill can fill in the city, county, and state as `San Francisco`, `San Francisco,` and `California` respectively. This level of autofill happens only if there is a unique match between the fields. For example, with San Francisco as the city and California as the state, autofilling the ZIP code would not work because San Francisco has multiple ZIP codes. This matching is done in the default autocompletion and autofill plugin, matching between the fields and the zone definitions.

Force an override of already filled fields

By default, if a field already has a value, PolicyCenter does not overwrite it. You can force it to override the field value with zone data by using a method on `AddressAutocompleteUtil` to autofill the address.

In addition, there are two PCF widgets that you can use to support autofill:

- `AddressAutoFillRange`
- `AddressAutoFillInput`

Both widgets provide a small icon for autofilling the fields on demand.

Add address autocomplete

Procedure

1. Create an `AddressAutocompleteHandler` instance in the appropriate PCF file.

   You create the instance by calling the factory method `gw.api.contact.AddressAutocompleteHandler.createHandler` with the following arguments:

   - A `String` identifying the field to complete
   - A comma-separated list of the reference fields, field names of the `Address` entity, that the application passes to the handler
   - A `Boolean` value that specifies whether PolicyCenter waits for a key press before fetching suggestions

2. Edit the PCF file in Guidewire Studio and add the factory method in the **Variables** tab of the PCF widget, calling the method in the `initialValue` field.

   Suppose that you want to construct an `AddressAutocompleteHandler` called `cityhandler` in the `GlobalAddressInputSet.default.pcf` file. In that case, you need to first add the `cityhandler` variable in the **Variables** tab, then set its various fields as follows:

   initialValue

   ```
   contact.AddressAutocompleteHandler
          .createHandler("City","City,County,State,PostalCode,Country",true)
   ```

   name

   ```
   cityhandler
   ```

recalculateOnRefresh

`false`

From the method call, you can see that this handler operates on the `City` field. The handler requires the following set of reference fields: `City`, `County`, `State`, `PostalCode`, and `Country`.

3. Edit the widget that uses the handler to specify which handler is responsible for autofill in the `autoComplete` property and the arguments the handler expects in the `autoCompleteArgIds` field.

   For example, for **AddressAutoFillInput: City**:

   action

   ```
   gw.api.contact.AddressAutocompleteUtil
         .autofillAddress(addressOwner.AddressDelegate,"City")
   ```

   actionAvailable

   `addressOwner.AutofillIconEnabled`

   autoComplete

   `cityhandler`

   autoCompleteArgIds

   `City,County,State,PostalCode,Country`

   available

   `addressOwner.isAvailable(fieldId.CITY)`

   This example uses an `AddressAutoFillInput` widget for the city field. This widget displays an autofill icon beside the field that acts as a cue. You do not have to use this specific type of widget for this purpose.

Address automatic completion and autofill plugin

The address automatic completion plugin uses interface `IAddressAutocompletePlugin` to declares methods that support automatic completion and fill-in for PolicyCenter. In the base configuration, Guidewire provides Java class `DefaultAddressAutocompletePlugin` as the implementation class for this plugin.

This class supports the use of the `address-config.xml` and `zone-config.xml` files. If you want to implement your own autofill and autocompletion configuration, such as one that does not use `zone-config.xml`, you can create a class that implements `IAddressAutocompletePlugin` and provide your own specialized logic.

Alternatively, you can extend `DefaultAddressAutocompletePlugin` and override the following methods, for example.

`autofillAddress(address : AddressFillable, triggerFieldName : String, doOverride : boolean)`

The purpose of this method is to autofill and address if there is a unique match with the entered data.

`getStates(country : Country): States[]`

This method retrieves the states for the given country.

Example: Add a country with a new address field

Basic configuration of the suburb field

This example shows how to add a new New Zealand address field named Suburb to PolicyCenter and ContactManager. The example configures classes, entities, and configuration files that support address fields so the new field can be used in the user interface and be available for autofill and autocompletion.

This example is not a full example showing how to perform a complete localization configuration, but rather focuses on configuring the new address field to work in ContactManager and PolicyCenter. After configuration, if you edit a contact's address and set the country to New Zealand, you will be able to use the Suburb field.

Note: Except where the instructions say explicitly to work only in ContactManager or only in PolicyCenter, do everything in both ContactManager and PolicyCenter.

This example includes the following steps to perform the basic configuration in Guidewire Studio™.

1. "Add a suburb field to the GlobalAddress entity" on page 117
2. "Make changes so suburb field uses autofill" on page 117
3. "Add a New Zealand locale type and suburb zone type" on page 118
4. "Add New Zealand localization configuration files" on page 119
5. "Add a New Zealand folder under the geodata folder" on page 119
6. "Add a New Zealand field validator file" on page 121
7. "Add typecodes for currency and jurisdiction" on page 121
8. "Configure the Currency XML file for New Zealand currency" on page 122
9. "Edit supporting user interface files and add New Zealand suburb field" on page 122
10. "Add a New Zealand suburb field to the user interface" on page 123
11. "Restart Studio and modify ContactManager" on page 124

Add a suburb field to the GlobalAddress entity

Procedure

1. Navigate to **configuration→config→Extensions→Entity** and double-click `GlobalAddress.etx` to open it in the editor.
2. Right-click the top element on the left, the **delegate(extension)** `GlobalAddress`, and choose **Add new→column**.
3. Enter the following values on the right:
   - `name` – `Suburb_Ext`
   - `type` – `varchar`
   - `nullok` – `true`
   - `desc` – `Address field for countries that have both a suburb and a city field`

   Guidewire recommends that you always add the `_Ext` extension to any new entity field that you add to the base PolicyCenter data model.
4. On the left, right-click the new `Suburb_Ext` column and choose **Add new→columnParam**.
5. Enter the following values on the right:
   - `name` – `size`
   - `value` – `50`

Next steps

The next step is "Make changes so suburb field uses autofill" on page 117.

Make changes so suburb field uses autofill

Before you begin

Complete the step "Add a suburb field to the GlobalAddress entity" on page 117 before you perform this step.

Procedure

1. Press `Ctrl+N` and enter `AddressFillableExtension`, and then press `Enter` to open this class.
2. Add the following lines of code:

```
property get Suburb() : String
property set Suburb(value : String)
```

3. Press `Ctrl+N` and enter `AddressFillableExtensionImpl`, and then press `Enter` to open this class.
4. Add the following line of code:

```
var _Suburb : String as Suburb
```

5. Press `Ctrl+N` and enter `UnsupportedAddressFillable`, and then double-click `UnsupportedAddressFillable` in the search results to open this class.
6. Add the following lines of code:

```
override property get Suburb(): String {
  return _unsupportedBehavior.getValue<String>()
}
override property set Suburb(value: String) {
  _unsupportedBehavior.setValue("Surburb")
}
```

7. Press `Ctrl+N` and enter `AddressAutofillableDelegate`, and then press `Enter` to open this class.
8. Add the following lines of code:

```
override property set Suburb(value : String) { _af.Suburb = value }
override property get Suburb() : String { return _af.Suburb }
```

9. Press `Ctrl+N` and enter `AddressEntityDelegate`, and then press `Enter` to open this class.
10. Add the following lines of code:

```
override property get Suburb() : String {
  return _ao.Address.Suburb }
override property set Suburb(value : String) {
  _ao.Address.Suburb = value }
```

Next steps

The next step is "Add a New Zealand locale type and suburb zone type" on page 118.

Add a New Zealand locale type and suburb zone type

Before you begin

Complete the step "Make changes so suburb field uses autofill" on page 117 before you perform this step.

Procedure

1. Navigate in the **Project** window to **configuration→config→Metadata→Typelist** and right-click `LocaleType.tti`.
2. Choose **New→Typelist extension** to create the file `LocalType.ttx`.
   - If you see a **Typelist Extension** dialog enabling you to choose one of several `LocaleType.ttx` locations, choose the one in `configuration/config/extensions/typelist/` and then click OK.
   - If you see a message saying that typelist extension is not allowed, then the file `LocaleType.ttx` already exists. Open that file instead.
3. In the editor for `LocaleType.ttx`, right-click the top element on the left, **typelistextension** `LocaleType`, and choose **Add new→typecode**.
4. Enter the following values for the new typecode:
   - `code` – `en_NZ`
   - `name` – `New Zealand (English)`
   - `desc` – `New Zealand (English)`
5. Navigate in the **Project** window to **configuration→config→Extensions→Typelist** and double-click `ZoneType.ttx`.
6. In the editor, right-click the top element on the left, **typelistextension** `ZoneType`, and choose **Add new→typecode**.

7. Enter the following values for the new typecode:
   - `code` – `suburb`
   - `name` – `Suburb`
   - `desc` – `Suburb`
8. In the editor right-click the `suburb` typecode and choose **Add new→Add categories**.
9. Click **suburb** in the list of typecodes and then click **Next**.
10. In the **Typelist** list, click `Country`, and in the **Category** list, click `NZ`, and then click **Finish**.

Next steps

The next step is "Add New Zealand localization configuration files" on page 119.

Add New Zealand localization configuration files

Before you begin

Complete the step "Add a New Zealand locale type and suburb zone type" on page 118 before you perform this step.

Procedure

1. Navigate in the **Project** window to **configuration→config→Localizations**.
2. Right-click **Localizations** and choose **New→File**. Name the new file `localization_en_NZ.xml`.
3. Copy the contents of `localization_en_US.xml` and paste it to this new file.
4. In `localization_en_NZ.xml`, configure the `<GWLocale>` element to use `en_NZ` as the country and to use date configurations appropriate for New Zealand:

```
<GWLocale code="en_NZ" name="New Zealand (English)" typecode="en_NZ">
  <DateFormat long="E d MMM yyyy" medium="d MMM yyyy" short="dd/MM/yyyy"/>
  <TimeFormat long="h:mm:ss a z" medium="hh:mm:ss a" short="h:mm a"/>
  <NumberFormat decimalSymbol="." thousandsSymbol=","/>
  <CurrencyFormat negativePattern="($#)" positivePattern="$#" zeroValue="-"/>
</GWLocale>
```

Next steps

The next step is "Add a New Zealand folder under the geodata folder" on page 119.

Add a New Zealand folder under the geodata folder

Before you begin

Complete the step "Add New Zealand localization configuration files" on page 119 before you perform this step.

About this task

In this step, you create a folder for New Zealand under the `geodata` folder and add configuration files for country, address, and zone.

Procedure

1. Navigate in the **Project** window to **configuration→config→geodata**.
2. Right-click **geodata** and choose **New→Package**. Name the new package `NZ`.
3. Copy the three files under the `geodata/AU` folder to this new folder. The three files are `address-config.xml`, `country.xml`, and `zone-config.xml`.
4. Double-click the copied `country.xml` file to open it in the editor.
5. Add `Suburb` to the `visibleFields` definition and remove `State`:

```
visibleFields=
  "Country,AddressLine1,AddressLine2,AddressLine3,Suburb,City,PostalCode"
```

6. Double-click the copied `address-config.xml` file to open it in the editor.
7. Set the file up to use `NZ` as the country and the name, and to use `Suburb` instead of `State`:

```
<AddressConfig xmlns="http://guidewire.com/address-config">
  <AddressDef name="NZ">
    <Match>
      <Field name="Country" value="NZ"/>
    </Match>
    <Fields>
      <Field name="City" zonecode="city">
          <AutoFillFromZone code="suburb"/>
          <AutoFillFromZone code="postalcode"/>
      </Field>
      <Field name="Suburb" zonecode="suburb">
      <!-- Do not autofill from the city because a definitive match
           to suburb from the city isn't possible-->
          <AutoFillFromZone code="postalcode"/>
      </Field>
      <Field name="PostalCode" zonecode="postalcode">
          <!-- Autofill first from suburb, because most people use suburb and
               postcode is a new concept. -->
          <!-- Postcodes are not unique by city. For example, 0420 maps to the
               following cities: Cable Bay, Coopers Beach, Mangonui, Taipa -->
          <AutoFillFromZone code="suburb"/>
          <AutoFillFromZone code="city"/>
          <ValidatorDef value="[0-9]{4}"
            description="Validator.PostalCode.NZ" input-mask="####"/>
      </Field>
      </Fields>
  </AddressDef>
</AddressConfig>
```

8. Double-click the copied `zone-config.xml` file to open it in the editor.
9. Change the country code to `NZ`:

```
countryCode="NZ"
```

10. Add `Suburb` to the zone definitions, change its relationship to postal code and city, and remove `State`.

    This file defines the CSV file order for each zone data entry to be postal code, suburb, city.

```
<Zones countryCode="NZ">
<!-- The uniqueness of the postcode is based on the containing zone.
     The postcode's container is the country and the deciding factor is whether
     or not the postcode is unique within the country. and not the fact that the same
     postal code is shared by multiple cities. For example, 0420 maps to the following
     cities: Cable Bay, Coopers Beach, Mangonui, Taipa -->
  <Zone code="postalcode" fileColumn="1" granularity="1" regionMatchOrder="1">
    <AddressZoneValue>Address.PostalCode.substring(0,4)</AddressZoneValue>
    <Links>
      <Link toZone="suburb" lookupOrder="1"/>
      <Link toZone="city" lookupOrder="2"/>
    </Links>
  </Zone>
  <Zone code="suburb" fileColumn="2" granularity="2" regionMatchOrder="2">
  <!-- As described for the postcode, the containing zone for the suburb
       is the city and therefore the suburb is not unique per city.
       Some examples: HillCrest is a suburb of both Hamilton and Rotorua.
       Avondale is a suburb of Auckland and Christchurch -->
    <ZoneCode> city + ":" + suburb </ZoneCode>
    <AddressZoneValue>
      Address.City + ":" + Address.Suburb
    </AddressZoneValue>
    <Links>
      <Link toZone="postalcode" lookupOrder="1"/>
      <Link toZone="city" lookupOrder="2"/>
    </Links>
  </Zone>
  <Zone code="city" fileColumn="3" granularity="3" regionMatchOrder="3">
  <!-- City (and town) names are unique in the country. They might have similar
       names such as Palmerston North, which is in the North Island,
       and Palmerston, which is in the South Island, but there is
       always a distinction. -->
```

```
        <AddressZoneValue> Address.City </AddressZoneValue>
      <Links>
        <Link toZone="postalcode" lookupOrder="1"/>
        <Link toZone="suburb" lookupOrder="2"/>
      </Links>
    </Zone>
  </Zones>
```

Next steps

The next step is “Add a New Zealand field validator file” on page 121.

Add a New Zealand field validator file

Before you begin

Complete the step “Add a New Zealand folder under the geodata folder” on page 119 before you perform this step.

About this task

In this step you add a field validator file for New Zealand and add the field validator for a New Zealand postal code to it.

Procedure

1. Navigate in the **Project** window to **configuration→config→fieldvalidators**.
2. Right-click **fieldvalidators** and choose **New→Package**, and then enter `NZ` and click **OK**.
3. Copy `fieldvalidators.xml` from **AU** to the new **NZ** folder.
4. In the new file, change `Validator.PostalCode.FourDigit` to `Validator.PostalCode.NZ`. The new validator definition is:

```
<ValidatorDef
  description="Validator.PostalCode.NZ"
  input-mask="####" name="PostalCode"
  value="[0-9]{4}"/>
```

Next steps

The next step is “Add typecodes for currency and jurisdiction” on page 121.

Add typecodes for currency and jurisdiction

Before you begin

Complete the step for “Add a New Zealand field validator file” on page 121 before you perform this step.

Procedure

1. Navigate in the **Project** window to **configuration→config→Extensions→Typelist**.
2. Double-click `Currency.ttx` to open this typelist in the editor.
3. In the editor, right-click the top element on the left, **typelistextension** `Currency`, and choose **Add new→typecode**.
4. Enter the following values for the new typecode:
   - `code` – `nzd`
   - `name` – `NZD`
   - `desc` – `New Zealand Dollar`
5. Double-click `Country.ttx` to open this typelist in the editor.
6. In the editor, right-click the `NZ` typecode and choose **Add new→Add categories**.

7. Click **NZ** in the list of category typecodes and then click **Next**.
8. In the **Typelist** list, click `Currency`, and in the **Category** list, click `nzd`, and then click **Finish**.
9. Double-click `Jursdiction.ttx` to open this typelist in the editor.
10. In the editor, right-click the top element on the left, **typelistextension** `Jursdiction`, and choose **Add new→typecode**.
11. Enter the following values for the new typecode:
    - `code` – `NZ`
    - `name` – `New Zealand`
    - `desc` – `New Zealand`
12. In the editor, right-click the `NZ` typecode and choose **Add new→Add categories**.
13. Click **NZ** in the list of category typecodes and then click **Next**.
14. In the **Typelist** list, click `Country`, and in the **Category** list, click `NZ`, and then click **Finish**.
15. In the editor, right-click the `NZ` typecode again and choose **Add new→Add categories**.
16. Click **NZ** in the list of typecodes and then click **Next**.
17. In the **Typelist** list, click `JurisdictionType`, and in the **Category** list, click `insurance`, and then click **Finish**.

Next steps

The next step is "Configure the Currency XML file for New Zealand currency" on page 122.

Configure the Currency XML file for New Zealand currency

Before you begin

Complete the step "Add typecodes for currency and jurisdiction" on page 121 before you perform this step.

Procedure

1. Navigate in the **Project** window to **configuration→config→currencies**.
2. Right-click **currencies** and choose **New→Package**, and then enter `nzd` and click **OK**.
3. Copy the `currency.xml` file from the **aud** folder to the new **nzd** folder.
4. In the editor for `nzd/currrency.xml`, change the currency code from `"aud"` to `"nzd"` and the currency description from `"Australian Dollar"` to `"New Zealand Dollar"`. The resulting code is:

```
<Currency xmlns="http://guidewire.com/currency">
  <CurrencyType code="nzd" desc="New Zealand Dollar" storageScale="2">
    <CurrencyFormat zeroValue="-"/>
  </CurrencyType>
</Currency>
```

Next steps

The next step is "Edit supporting user interface files and add New Zealand suburb field" on page 122.

Edit supporting user interface files and add New Zealand suburb field

Before you begin

Complete the step "Configure the Currency XML file for New Zealand currency" on page 122 before you perform this step.

Procedure

1. Press `Ctrl+Shift+N` and enter `Address.en`, and then double-click `Address.en` in the search results to open it in the editor.

2. In the top part of the editor, click the **Add** icon and then enter the following values:
   - **Name** – `suburb`
   - **Entity Path** – `Address.Suburb`
3. Below this top pane, there is a code pane with a `Default` tab. Click this tab and add a line of code defining a `Suburb` to the list of `formatter` address field definitions. If necessary, add a comma to the end of the preceding line:

```
:CEDEXBureau = CEDEXBureau,
:Suburb = suburb
```

4. In ContactManager only, this code pane also has a `Full` tab. Click this tab and make the same addition as you did in the `Default` tab.
5. Press `Ctrl+N` and enter `AddressOwnerFieldId`, and then double-click `AddressOwnerFieldId` in the search results to open this class in the editor.
6. Add the following line of code to the constant declarations for available fields:

```
public static final var SUBURB : AddressOwnerFieldId =
  new AddressOwnerFieldId("Suburb")
```

7. Add the `Suburb` field to the constant representing all PCF address fields:

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

8. Hide the `Suburb` field from the default search screens by changing the settings in `AddressOwnerFieldId` for `HIDDEN_FOR_SEARCH` and `HIDDEN_FOR_PROXIMITY_SEARCH`:

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

public final static var HIDDEN_FOR_PROXIMITY_SEARCH : Set<AddressOwnerFieldId> =
  { ADDRESSLINE1KANJI, ADDRESSLINE2KANJI, SUBURB, CITYKANJI,
    CEDEX, CEDEXBUREAU }.freeze()
```

   You can change this setting later if New Zealand is to be the default region.
9. Press `Ctrl+N` and enter `AddressFormatter`, and then double-click `AddressFormatter` in the search results to open this class in the editor.
10. Add the following `case` to `switch(_addrCountry)` to indicate the suburb, city, and postal code format for a New Zealand address:

```
case TC_NZ:
  append(cszBuf, fieldId.SUBURB, delimiter, addr.Suburb)
  append(cszBuf, fieldId.CITY,  " ", addr.City)
  append(cszBuf, fieldId.POSTALCODE, " ", addr.PostalCode)
  break
```

Next steps

The next step is “Add a New Zealand suburb field to the user interface” on page 123.

Add a New Zealand suburb field to the user interface

Before you begin

Complete the step “Edit supporting user interface files and add New Zealand suburb field” on page 122 before you perform this step.

Procedure

1. Navigate in the **Project** window to **configuration→config→Page Configuration→pcf→address** and double-click `GlobalAddressInputSet.default` to open it in the editor.

2. Click `InputSet: GlobalAddressInputSet` at the top of the screen to open its `Properties` pane at the bottom.
3. Click the **Variables** tab, and then click **Add** (plus sign) to add a new variable. Enter the following values:
   - **initialValue** –
     ```
     contact.AddressAutocompleteHandler.createHandler(
            "Suburb","Suburb,City,County,State,PostalCode,Country",true)
     ```
   - **name** – `suburbhandler`
4. In the screen, select the Address 3 `Input` widget.
5. Drag an `AddressAutofillInput` widget from the **Toolbox** to the Address 3 `Input` widget and drop it so it appears after this widget. Select the new widget.
6. In the **Properties** panel, set the following values:
   - **editable** – `addressOwner.isEditable(fieldId.SUBURB)`
   - **id** – `Suburb`
   - **required** – `addressOwner.isRequired(fieldId.SUBURB)`
   - **value** – `addressOwner.AddressDelegate.Suburb`
   - **action** –
     ```
     if (addressOwner.AutofillIconEnabled)
       gw.api.contact.AddressAutocompleteUtil.autofillAddress(
              addressOwner.AddressDelegate, "Suburb")
     ```
   - **actionavailable** – `addressOwner.AutofillIconEnabled`
   - **autoComplete** – `suburbhandler`
   - **autoCompleteArgIds** – `Suburb,City,County,State,PostalCode,Country`
   - **available** – `addressOwner.isAvailable(fieldId.SUBURB)`
   - **visible** – `addressOwner.isVisible(fieldId.SUBURB)`
   - **label** – `displaykey.Web.AddressBook.AddressInputSet.Suburb`

     Do the following to create this display key.
     - If you see a red icon:
       a. Click the red icon and choose **Create Display Key**.
       b. Under `en_US`, enter `Suburb` and then click **OK**.
     - If no red icon appears:
       a. Navigate in the **Project** window to **configuration→config→Localizations→Resource Bundle 'display'** and double-click `display.properties` to open this file in the editor.
       b. Add the key `Web.AddressBook.AddressInputSet.Suburb = Suburb` to that file.

Next steps

The next step is “Restart Studio and modify ContactManager” on page 124.

Restart Studio and modify ContactManager

Before you begin

Complete the step “Add a New Zealand suburb field to the user interface” on page 123 before you perform this step.

Procedure

1. Close and restart Guidewire Studio™ to clear any errors you see reported in the classes that you changed.
2. Modify the `ContactMapper` class in both PolicyCenter and ContactManager to pass the field to and from ContactManager:

a. Navigate in the **Project** window to **configuration→gsrc** and then to `gw/contactmapper/ab900/ContactMapper`.

b. Add a call to `fieldMapping` for `Address#Suburb` after the field mapping call for `Address#CEDEXBureau`:

```
fieldMapping(Address#CEDEXBureau),
fieldMapping(Address#Suburb),
```

3. Make the following web service change in ContactManager only:

a. If necessary, open Guidewire Studio for ContactManager.

b. Navigate in the **Project** window to **configuration→gsrc** and then to `gw/webservice/ab/ab900/abcontactapi/AddressInfo`.

c. Add the following variable definition to the list of variables at the top of the class:

```
public var Suburb : String
```

d. Add the following line of code to the method `construct(address : Address)`:

```
this.Suburb = address.Suburb
```

4. Restart ContactManager.

5. In Guidewire Studio for PolicyCenter, reload the ContactManager web service, as follows:

a. In the **Project** window, navigate to **configuration→gsrc→wsi→remote→ab→ab900** and double-click `ab900.wsc` to open it in the editor.

b. In the Resources pane, click `${ab}/ws/gw/webservice/ab/ab900/abcontactapi/ABContactAPI?wsdl` to select it.

c. At the bottom of the `Resources` pane, click **Fetch Updates**.

See also

- “Overview of global addresses” on page 106
- “Configuring address data and field order for a country” on page 108
- “Zone configuration” on page 98
- *Guidewire Contact Management Guide*

Additional information for configuring New Zealand localization

To test autofill and autocomplete in the New Zealand configuration, you need to import New Zealand sample data supporting the zones defined in `zone-config.xml`. If you use the previous `zone-config.xml` configuration, the format of the data is postal code, suburb, city. For example:

```
0110,Abbey Caves,Whangarei
9018,Abbotsford,Dunedin
3330,Acacia Bay,Taupo
8011,Addington,Christchurch
```

- For information on creating zone information in a file you can import, see “Zone configuration” on page 98.
- For information on importing the file you create, see the *System Administration Guide*.

Additional configurations you can perform are:

- Adding New Zealand jurisdictions to `Jursidiction.ttx` to provide drop-down lists of jurisdictions in the user interface. The previous example adds only New Zealand itself as a country typecode to this typelist.
- Adding New Zealand regions in the PolicyCenter Administration tab. You can then associate these regions with user groups and use them to assign work. See:
  - *Application Guide*
- Configuring New Zealand language support and regional formats. For example, after completing the previous configuration for the `Suburb` field, the only English language support is `en_US`. See:
  - "Working with languages" on page 19
  - "Working with regional formats" on page 67

chapter 18

Configuring and localizing phone information

Phone number formats 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.

See also

- "Gosu field validation" on page 142

Phone configuration parameters

PolicyCenter uses the following configuration parameters in `config.xml` to set phone-related information.

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

Configuring area codes and phone number validation

You can set area codes for North America, validate international phone numbers and area codes, and validate phone numbers and area codes with alternate formats in the phone configuration files.

Accessing phone configuration files

Guidewire stores phone configuration files in the following location in Guidewire Studio **Project** window:

configuration→config→phone

Configuration files in the phone 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.
PhoneNumberAlternateFormats.xml	Additional area codes and validation rules for international phone numbers. 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 PolicyCenter data subdirectory. To regenerate phone data, run the following gwb utility from the application installation directory:

```
gwb genPhoneMetadata
```

Working with PhoneNumberMetadata.xml

This configuration file specifies phone number formats for countries and possibly for territories. Each country phone number definition starts with the tag <territory id="*country_name*". For example, if you are configuring United States phone data, you can search for <territory id="US" to find that configuration.

PolicyCenter uses possibleNumberPattern under generalDesc for validation of phone numbers. PolicyCenter does not perform any of the stricter validations because the formats can change rapidly and a user can enter a mobile number into a home phone field.

The formatting controls are in availableFormats. They use standard regex capture groups. The only sections you need to configure are possibleNumberPattern and availableFormats.

See the comments at the beginning of the file and at the beginning of each <territory> section for more information.

Change default phone localization

About this task

For the United States (US), the area code is defined in PhoneNumberMetaData.xml to display as three numbers followed by a dash. For example, the area code 202 displays as 202-555-1234. You can change this default display for the US, Canada, and any other NANPA countries defined in this file that rely on the US format. For example, change the default to use parentheses, such as (202) 555-1234, as follows:

Procedure

1. In Guidewire Studio™, navigate in the **Project** window to **configuration→config→phone** and then double-click PhoneNumberMetaData.xml to open the file in the editor.
2. Find the entry for the United States by searching for:

```
territory id="US"
```

3. In that element are two numberFormat elements. The second of these elements defines the pattern for parsing that includes an area code. It includes a <format> element that defines the format of the string.

```
<numberFormat pattern="(\d{3})(\d{3})(\d{4})">
  <format>$1-$2-$3</format>
```

4. Change this <format> element to use parentheses and a space, as follows:

```
<format>($1) $2-$3</format>
```

5. Open a command prompt from the PolicyCenter installation directory and run the following utility:

```
gwb genPhoneMetadata
```

6. Restart ClaimCenter to pick up this change.

Phone number data model

In Guidewire PolicyCenter, the Contact entity has the following fields that support phone numbers:

- `PrimaryPhone`
- `FaxPhone`
- `FaxPhoneCountry`
- `FaxPhoneExtension`
- `HomePhone`
- `HomePhoneCountry`
- `HomePhoneExtension`
- `WorkPhone`
- `WorkPhoneCountry`
- `WorkPhoneExtension`

The Person entity inherits all these fields, and, in addition, has the following fields that support phone numbers:

- `CellPhone`
- `CellPhoneCountry`
- `CellPhoneExtension`

For the listed entity fields:

- The phone country fields are typekeys that reference the `PhoneCountryCode` typelist, which provides the list of ISO country codes for phone data.
- The extension fields define `varchar` columns.

These two types of fields are associated with the main phone column through the following `columnParam` subelements:

- `phonecountrycodeProperty`
- `extensionProperty`

For example, in the XML text of the `Contact.eti` file, notice the following definitions.

```
<typekey desc="Home phone country." name="HomePhoneCountry" nullok="true"
      typelist="PhoneCountryCode"/>

<column desc="Home phone number associated with the contact." name="HomePhone"
        nullok="true" type="phone">
  <columnParam name="phonecountrycodeProperty" value="HomePhoneCountry"/>
  <columnParam name="extensionProperty" value="HomePhoneExtension"/>
</column>

<column desc="Home phone extension." name="HomePhoneExtension" nullok="true" type="varchar">
  <columnParam name="size" value="60"/>
</column>
```

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 the following:

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

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

Edit mode

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

  Note: The user-entered value of an extension can be no longer than seven characters. The length is enforced in the validator expression for the field.

Read-only mode

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.

PhoneFields interface

PolicyCenter interface `gw.api.phone.PhoneFields` is a representation of a phone number object. It contains properties for the following items:

- `CountryCode`
- `NationalSubscriberNumber`
- `Extension`
- `PhoneType`

In the base PolicyCenter, Guidewire provides class `ContactPhoneDelegate` as the default implementation class for the `PhoneFields` interface. This class wraps the three phone columns into a `ContactPhoneDelegate` object if provided with a `Contact` object and a `PhoneType` in the constructor. `ContactPhoneDelegate` sets the primary phone value for the contact.

PhoneOwner interface

The interface `gw.api.phone.PhoneOwner` defines behavior for the phone number widget, such as availability, editability, visibility, and targeted post-on-change behavior. The implementations in the base configuration are `StandardPhoneOwner` and `BusinessPhoneOwner`. The main difference between the two classes is that `BusinessPhoneOwner` provides an extra field for the phone extension.

Phone numbers in edit mode

International phone numbers begin with a country code and use the following format:

```
+phoneCountryCode phoneNumber extensionNumber
```

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 chose. For PolicyCenter to show the **Region Code** drop-down list, 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 format
US	US	Domestic United States
US	FR	International French
FR	FR	Domestic French
FR	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

- *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** (gear icon) menu. A user's selection for default phone region overrides the default phone region set for the server.

Configure the phone extension read-only label

About this task

If the user previously entered an extension for a phone number, in read-only mode you see a text label preceding the extension number. In the base configuration, Guidewire defines this label to be the letter `x`. For example, a San Francisco, California phone number with extension 123 displays in the detail view for a contact as the following number:

415-555-1212 x123

Display property `Java.PhoneUtil.FormatPattern.Extension` (in file `display.properties`) defines this label. In the base configuration, Guidewire defines this property for United States English as `{0} x{1}`. It is possible to change the extension label by changing the property definition.

Procedure

1. Open Guidewire Studio for PolicyCenter.
2. In the Studio **Project** window, navigate to the following location:

   configuration→config→Localizations→Resource Bundle 'display'
3. Double-click file `display.properties` to open the file in the editor.
4. Change the value of `Java.PhoneUtil.FormatPattern.Extension` to the desired value.

   For example, change the value to the following:

   ```
   {0} Ext.{1}
   ```

Result

If you make the suggested change, and, if the display language is United States English, the example phone number displays as follows in read-only mode:

415-555-1212 Ext.123

See also

- “Display keys and localization” on page 31
- “Localizing display keys” on page 35

Converting phone numbers from previous formats

PolicyCenter provides a plugin interface, `IPhoneNormalizerPlugin`, that you can use to customize the conversion of phone numbers from a format that does not match the format used in PolicyCenter. The plugin is called by the Phone Number Normalizer work queue. The plugin is also called whenever an entity containing a phone number is modified in PolicyCenter or restored from the archive.

See also

- “Phone number normalizer plugin” in the *Integration Guide*
- “Phone Number Normalizer work queue” in the *System Administration Guide*

chapter 19

Linguistic search and sort

You can 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: 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 the *Configuration Guide*.

Understanding 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_`*`LanguageCode`*`.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 language code. You define and manage language characteristics in `language_`*`languageCode`*`.xml` files. You define these files for each localization file, such as `language_en_US.xml`. To access the localization folders, open Guidewire Studio™ and navigate in the **Project** window to **configuration→config→Localizations**.

Each `language_`*`languageCode`*`.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 `SortCollation` `strength` attribute value depends on the specific language.
- `<LinguisticSearchCollation>` – Element `<LinguisticSearchCollation>` supports a `strength` attribute that you use to define the searching collation strength for a language.

See also

- For information on `<LinguisticSearchCollation>`, see "Configuring Oracle search in language_languageCode.xml" on page 135.
- For information on `<SortCollation>`, see "Configuring database sort in language_languageCode.xml" on page 138.
- "Configuring search in the PolicyCenter database" on page 135
- "Configuring sort in the PolicyCenter database" on page 138

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 and supports case insensitive searches, regardless of which DBMS you use.

Note: Setting this attribute to `true` on a column makes it possible to do case-insensitive search on a denormalized version of the column. Setting it to `false` enables binary search on the column itself, as described in "Linguistic search and sort" on page 133. Setting this attribute does not affect language or locale for searches. Using locale in searching and collation order are controlled by the configuration of the user's locale and the strength of the `LinguisticSearchCollation` entry in the `language_`*`languageCode`*`.xml` file.

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

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

See also

- For information on the `<column>` element and the attributes that you can set on it, see the *Configuration Guide*.
- For information on `LinguisticSearchCollation`, see:
  - "Searching and sorting in configured languages" on page 134
  - "Configuring Oracle search in language_languageCode.xml" on page 135
  - "General search rules" on page 137

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 136 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_languageCode.xml

Guidewire provides the ability to configure language-appropriate linguistic search capabilities through the `<LinguisticSearchCollation>` element. This subelement of `<GWLanguage>` is defined in `language_`*`languageCode`*`.xml`. You use the `strength` attribute of this subelement to configure and control specialized search behavior.

IMPORTANT Any change to the `<LinguisticSearchCollation>` element in `language_*languageCode*.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_languageCode.xml" on page 138.
- 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`	• accent-insensitive • case-insensitive
`secondary`	• accent-sensitive • case-insensitive

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_ja.xml` file is an example for Japanese with suggested settings.

```
<?xml version="1.0" encoding="UTF-8"?>
<Language xmlns="http://guidewire.com/language">
  <GWLanguage code="ja" name="Japanese" typecode="ja">
    <LinguisticSearchCollation strength="primary"/>
    <SortCollation strength="primary"/>
  </GWLanguage>
</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 `collations.xml`. To access `collations.xml`, open Guidewire Studio™ and navigate in the **Project** window 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 `GWNormalize` Java class directly to meet your business needs. It is useful to modify the `GWNormalize` 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.
- 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.

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 `strength` attribute on `<LinguisticSearchCollation>` is set to `primary`, meaning accent-insensitive, ignore Japanese small/large letter differences in Katakana or Hiragana. Searches in which this attribute is set to `secondary` 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 *kana-insensitive* searching.
Long dash (—)	All searches in Japanese ignore the long dash character.
Sound marks (`` and °)	All searches in Japanese in which the `strength` attribute on `<LinguisticSearchCollation>` is set to `primary` ignore sound marks. Searches in which this attribute is set to `secondary` 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 `collations.xml`.

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: • ä and ae • ö and oe • ü and ue
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 `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_`*`languageCode`*`.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 `language_`*`languageCode`*`.xml` file for the language.

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.

Configuring database sort in language_languageCode.xml

For optional use, the `<SortCollation>` subelement of the `<GWLanguage>` element in `language_`*`languageCode`*`.xml` controls specialized sorting behavior. To access the localization files, open Guidewire Studio™ and navigate in the **Projects** window to **configuration→config→Localizations**.

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_*LanguageCode*.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_de_DE.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"/>
...
```

Collation in QuickStart (H2 development) database

To specify collation in file collations.xml for an H2 database, use a standard locale name format, for example de_DE for Germany, ja_JP for Japan.

Determining the order of typekeys

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

If there is no `.sort` file defined for the typelist and language:

- PolicyCenter uses the priority associated with each typekey in its typelist to order the typekeys by priority order.
- For typekeys with the same priority, PolicyCenter applies the language collation rules to the typekey display names.

If there is a `.sort` file defined for the typelist and language:

- PolicyCenter uses the order of the typekeys specified in that file.
- For typekeys from the typelist that are not defined in the `.sort` file, PolicyCenter orders them after the defined typekeys, applying the language collation rules to these 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_`*`LanguageCode`*`.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 a localized sort order for localized typecodes” on page 38.
- For information on setting typecode priority, see the *Configuration Guide*.

chapter 20

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.

See also

- *Configuration Guide*

About 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 various `fieldvalidtors.xml` files under the `fieldvalidators` folder:

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

See also

- “Localizing field validators for national field validation” on page 142
- *Configuration Guide*

Localizing field validators for national field validation

About this task

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

Procedure

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.

Gosu field validation

An advanced kind of field validator defines a Gosu class that handles field validation programmatically. You can develop Gosu classes that act as field validators. PolicyCenter provides a class framework for developing Gosu-based field validators, located in the `gw.api.validation` package. Gosu-based field validators must extend the abstract `FieldValidatorBase` class.

Enabling national field validation for phone data

PolicyCenter uses the Gosu class `gw.api.validation.PhoneValidator` as the default mechanism to validate phone number correctness.

To enable the new `PhoneValidator` validation functionality, you configure `fieldvalidators.xml` with the fully qualified name of the Gosu class. For example:

```
<ValidatorDef description="Validator.Phone"
              name="LocalizedPhoneValidator"
              validation-type="gosu"
              value="gw.api.validation.PhoneValidator"/>
```

As shown in the previous example, you must set the validation type to `gosu`.

The validator does not trigger validation unless the associated phone country is set. This trigger functionality provides backwards compatibility with old data.

See also

- “Configuring and localizing phone information” on page 127