



# Atomic Widgets



January 21, 2014

© Guidewire Software, Inc. 2001-2014. All rights reserved.  
Do not distribute without permission.

Guidewire training materials contain Guidewire proprietary information that is subject to confidentiality and non-disclosure agreements. You agree to use the information in this manual solely for the purpose of training to implement Guidewire software solutions. You also agree not to disclose the information in this manual to third parties or copy this manual without prior written consent from Guidewire. Guidewire training may be given only by Guidewire employees or certified Guidewire partners under the appropriate agreement with Guidewire.

# Lesson objectives

- By the end of this lesson, you should be able to:
  - Create atomic widgets
  - Bind an atomic widget to the data model
  - Create and modify widget labels
  - Use optional widget properties

This lesson uses the notes section for additional explanation and information.  
To view the notes in PowerPoint, select View → Normal or View → Notes Page.  
When printing notes, select Note Pages and Print hidden slides.

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

2

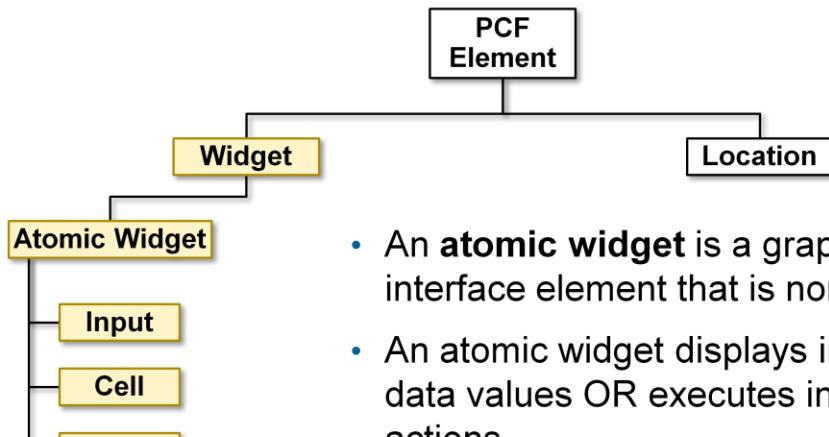
G U I D E W I R E



## Lesson outline

- Atomic widget fundamentals
- Creating atomic widgets
- Binding widgets to the data model
- Widget labels and display keys
- Optional widget properties
- Deploy the PCF and display keys

# Atomic widgets



- An **atomic widget** is a graphical user interface element that is non-divisible
- An atomic widget displays individual data values OR executes individual actions
- Use for input and cell widgets to bind data values
  - Data values often associated with root objects, query objects, or related objects

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

4

GUIDEWIRE

A widget is a graphical user interface element responsible for interacting with the user. "Widget" is a Java specific, industry term. Widgets maintain and draw their state using some combination of graphical drawing operations. Using the mouse or the keyboard, the user can change the state of a widget. When a state change occurs, whether initiated by the user or the application code, widgets redraw to show the new state.

Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Atomic Widget is also a conceptual representation. There is no <AtomicWidget /> element.

A location is a place to which you can navigate in the interface. Locations are used primarily to provide a hierarchical organization of the interface elements, and to assist with navigation. PCF elements that are considered locations include pages, wizards, worksheets, forwards, and location groups.

A widget is a PCF element that a Guidewire application renders into an HTML object. Buttons, menus, text boxes, and dropdown lists are all examples of PCF elements that are considered widgets. Some widgets that are not visually presented in the web browser, but are rendered as an HTML object, such as a hidden input.

# Atomic widgets: examples

- Button
  - Use to execute actions, typically in toolbars
- Cell
  - Use to display and edit data in list views
- Input
  - Use to display and edit data in detail views
- Layout (see notes)

The screenshot shows a software application window titled "Addresses". At the top, there is a toolbar with buttons for "Update", "Cancel", "Add", "Remove", and "Delete Secondary Addresses". Below the toolbar, there is a list view tab labeled "Address Detail" which is currently selected. This tab contains fields for "Address Type" (set to "Home"), "Description" (set to "Home"), "Country" (set to "United States"), "Address 1" (set to "345 Fir Lane"), "Address 2" (empty), "Address 3" (empty), "City" (set to "La Canada"), "County" (empty), "State" (set to "California"), and "ZIP Code" (set to "91352"). To the right of the "Address Detail" tab, there is another tab labeled "Addresses" which is currently active. This tab displays a list of addresses with columns for "Primary", "Address Type", and address details. One address in the list is highlighted, showing "Home" as the address type and "345 Fir Lane, La Canada, ..." as the address details. A red box highlights the "Update" button in the toolbar, and another red box highlights the "Home" address type in the list view.

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

5

G U I D E W I R E

Layout widgets help organize the user interface to improve usability and legibility. Common layout widgets include the label widget and the input divider widget. For label widgets, you specify a value for the label property. For input divider widgets, there are no properties to specify.



## Lesson outline

- Atomic widget fundamentals
- Creating atomic widgets
- Binding widgets to the data model
- Widget labels and display keys
- Optional widget properties
- Deploy the PCF and display keys

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

6

G U I D E W I R E

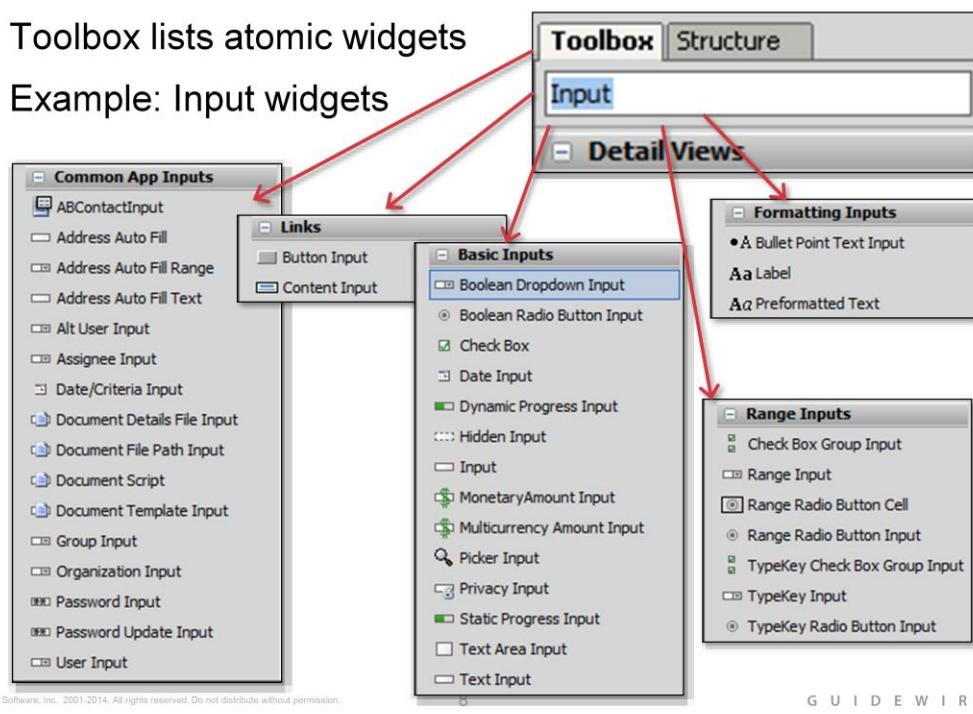
## Steps to create an atomic widget

1. Select the widget from the toolbox
2. Add the widget to the PCF
3. Specify the required properties
  - a) id property is unique name for widget0
  - b) value property is for widget that display data
4. Specify a display key for the label property
5. Define additional properties specific to the widget type
6. Deploy the PCFs and display keys

Guidewire recommends using "Ext" prefix for display keys. In a given widget property, specify the display key name. If necessary, localize display key values.

## Step 1: Select the widget

- Toolbox lists atomic widgets
- Example: Input widgets



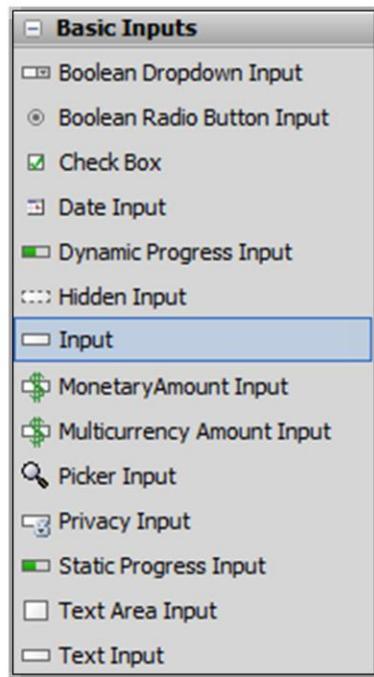
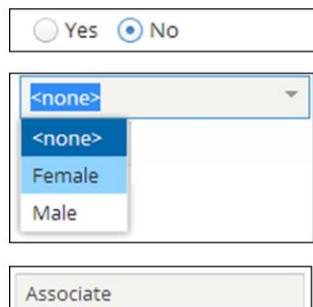
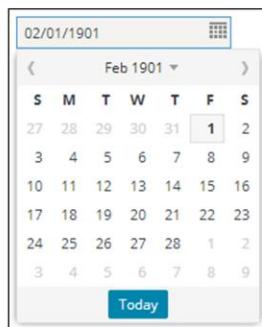
Atomic widgets are created by adding a widget into a container widget, such as a detail view panel, list view panel, or toolbar.

Widgets are listed in the Toolbox tab, which appears on the right side of the PCF Editor. Initially, all widgets are listed. You can, however, enter a string into the filter field, which causes the Toolbox to list only widgets whose descriptions contain that string. To view a widget tool description, mouse over the name of the widget tool in the Toolbox. The description appears in a tooltip.

Since PCF files are XML documents, each widget in the toolbox represents a specific XML element.

# Generic input widget

- Dynamically determines input type based on value data type
  - Boolean → Boolean Radio Input
  - Date → Date Input
  - TypeKey → TypeKey Input
  - Text → TextInput



© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

9

G U I D E W I R E

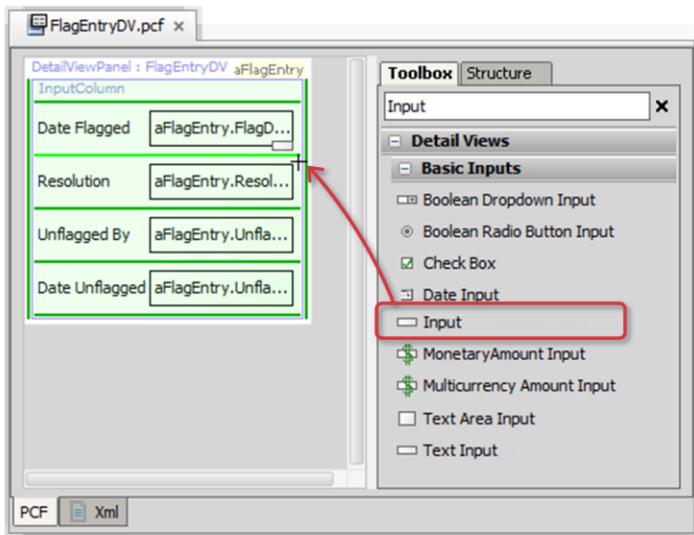
The generic input widget can be bound to any value. The type of the value widget (e.g., textbox, dropdown or checkbox) will be determined dynamically based on the value type, for example:

- Boolean - BooleanRadioInput
- Date - DateInput
- TypeKey - TypeKeyInput

If no such mapping is found, the default is to use a TextInput.

It may seem negligible, but to resolve the underlying data type for the widget value takes extra application processing. In aggregate, widget resolution can be a performance issue. Wherever possible, use the input widget that best matches the needs of the user, widget special properties, and the underlying data type value.

## Step 2: Add the widget to the PCF



- Light green line - current place where new widget will go
- Dark green line - places where new widget can go

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

10

G U I D E W I R E

To add a widget, click its name in the Toolbox and hold the mouse cursor down. As you begin to drag the widget, Studio changes the mouse cursor so that it includes the icon for that widget. Studio places a green line on the canvas at every location on the canvas that it is possible to place the widget. Studio highlights the green line that is nearest on the canvas to the cursor. Studio also overlays in green the element containing the highlighted green line

## Step 3: Specify the required properties

The screenshot shows the Guidewire PCF editor interface. On the left is the Editor canvas with a panel titled 'FlagEntryDV.pcf' containing several input fields: 'Date Flagged', 'Reason', 'Resolution', 'Unflagged By', and 'Date Unflagged'. The 'Reason' field is highlighted with a red box. On the right is the Structure tab of the Toolbox, showing a tree view of the PCF structure with nodes like 'DetailViewPanel : FlagEntryDV', 'Variable', 'InputColumn', 'Input : DateFlagged', 'Input : Reason', etc. Below the Structure tab is the Properties window. It has tabs for 'PCF' and 'Xml', and is currently on the 'Properties' tab. A specific entry, 'Input: Reason', is selected and highlighted with a red box. In the Properties grid, under the 'Basic properties' section, the 'id\*' property is set to 'Reason' (highlighted with a yellow background), and the 'value\*' property is set to 'aFlagEntry.Reason' (also highlighted with a yellow background). Other properties shown include 'editable' (set to 'false'), 'label' (set to 'displaykey.Training.Reason'), and 'required'.

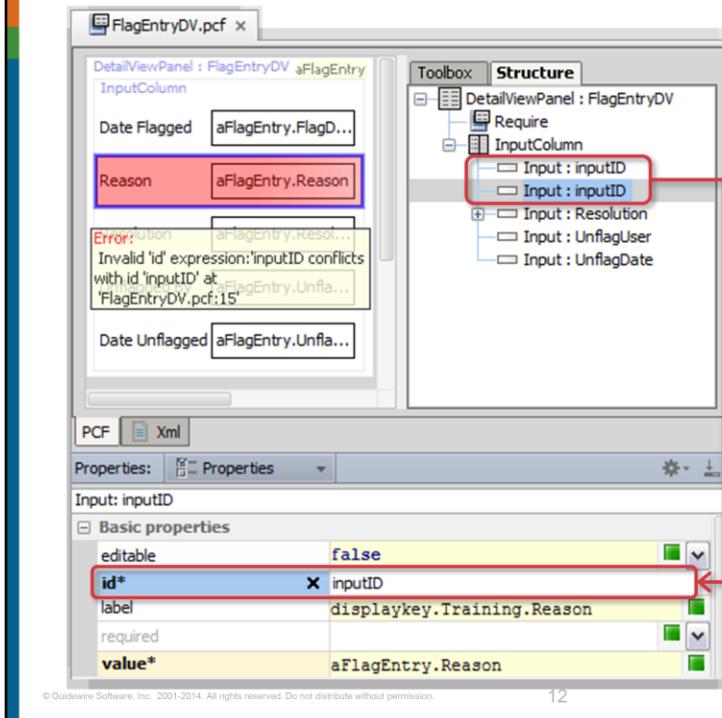
All PCF elements have definable properties in the Properties window. To view properties of a PCF file, click its title link in the upper-left corner. To view properties of any element, click that element.

The Properties window contains multiple property tabs. Click a tab to edit the associated properties. Some properties are not editable. Other properties are required. Required properties have an asterisk and the property name appears against a yellow background.

If you select a property, variable, or entry point, an "X" icon appears on the right-hand side of the cell for that property, variable, or entry point. You can click the "X" to restore the selected property, variable, or entry point to its default value.

The Properties window validates each property expression and/or and value.

## Step 3a: Specify the ID property



- PCF validation will display error in canvas when two widgets have the same value for an id property
- id\* property must be unique within a given PCF
  - id property not required for all types of widgets

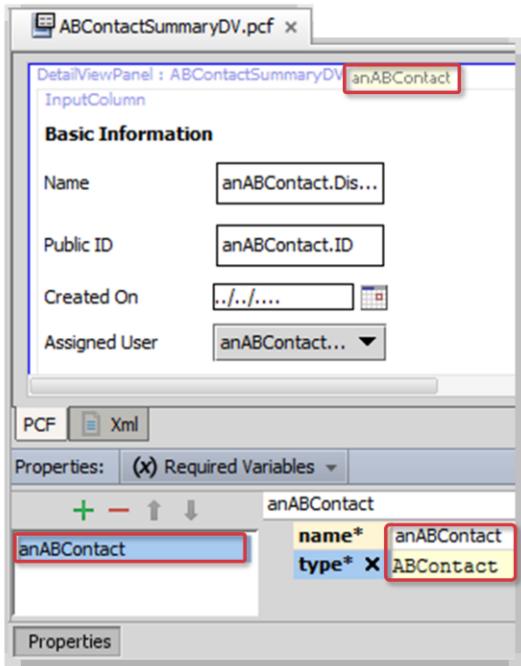
Most atomic widgets require IDs. However, some widgets, such as labels and dividers, do not require IDs.



## Lesson outline

- Atomic widget fundamentals
- Creating atomic widgets
- Binding widgets to the data model
- Widget labels and display keys
- Optional widget properties
- Deploy the PCF and display keys

# Input and cell widgets bind data



- Input and cell atomic widgets often bind to data
  - From variable object(s) defined in parent container
- Object data can be
  - Data backed (database)
  - Virtual property
- Required Variables tab
  - Defines data object variable name and type
  - Example:  
anABContact is of type ABContact



© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

14

G U I D E W I R E

Most container widgets have at least one required object that contains data fields. One way to think of this is that there is at least one root object for a given container.

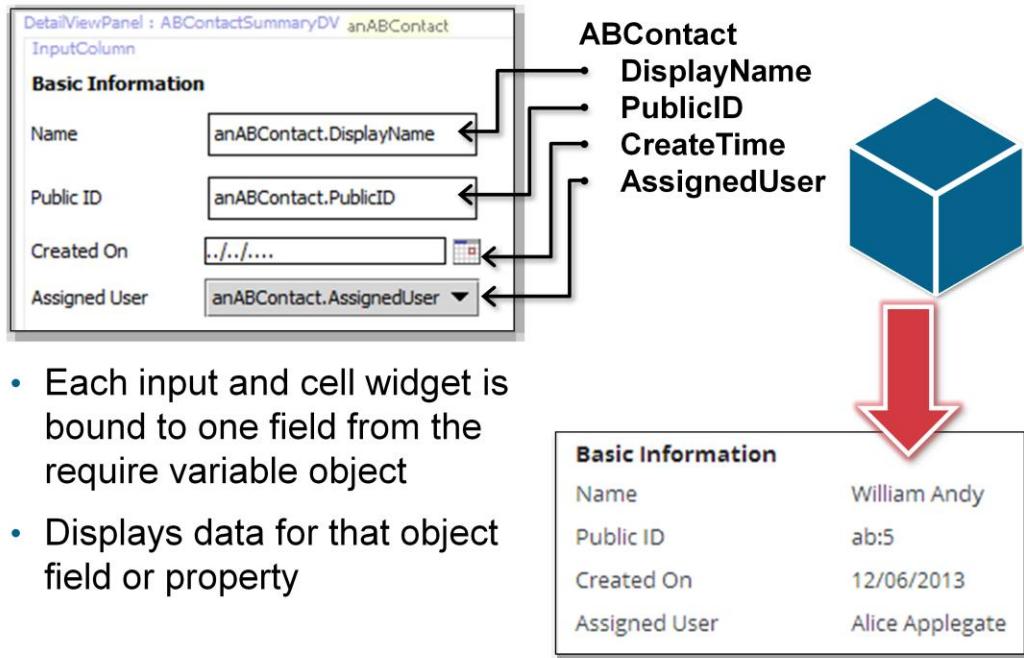
It is possible to have more than one defined object as it is also possible to not have a required object at all.

In later lessons, you will learn more about defining the required variable object and type.

There are container widgets that do not have any required objects such as containers that display only static information. For example, the detail view on the NoContact page has no associated object. If a user clicks the Contact tab without ever having searched for a contact, they will navigate to the NoContact page. The detail view contains a single static label that states "You have not yet viewed any contacts. To view a contact, you must first search for it on the Search tab."

Search screens typically have no required object and display no data initially. If you execute a search, then the data in the search results is displayed. This data comes from the search itself, though, and not from any required variable object.

# Each data widget is tied to one field (1)

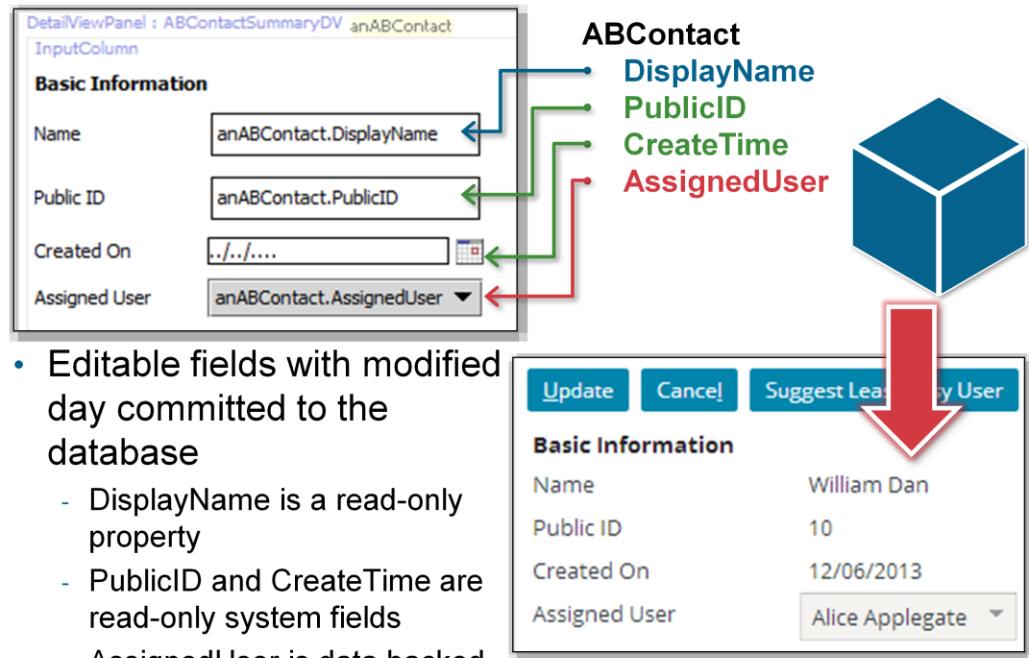


© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

15

G U I D E W I R E

## Each data widget is tied to one field (2)



16

G U I D E W I R E

If the root object field is editable and the widget is enabled to be editable, then changed data is committed to the database. In the slide example, only AssignedUser is an editable, data backed field for the anABContact required variable. DisplayName is an internal property for entity names. PublicID and CreateTime are read-only fields.

# Dot notation and subtype casting

- PCF specifies object variable datatype

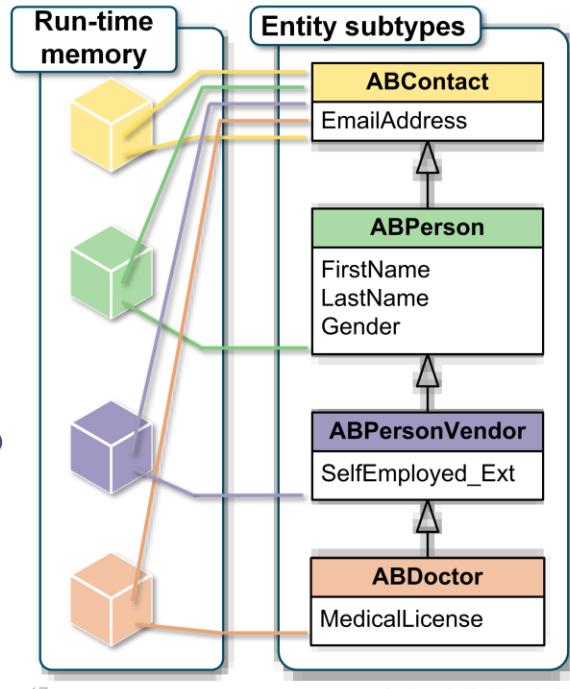
```
var anABContact : ABContact  
anABContact.EmailAddress
```

- Cast subtype reference

```
(anABContact as ABPerson)  
.Gender
```

```
(anABContact as ABPersonVendor)  
.SelfEmployed_Ext
```

```
(anABContact as ABDoc)  
.MedicalLicense
```



© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

17

G U I D E W I R E

When a Guidewire application retrieves data from the database, it stores that data as an object in runtime memory. It works with the in-memory information until the data needs to be committed or re-retrieved from the database.

When an object is created, all of the information about the object is copied into memory. If the object is subtyped, then data for all the fields at every relevant subtype level is retrieved. For example, if an object is an ABDoc, then information is copied over for fields at the ABContact, ABPerson, ABPersonVendor, and ABDoc levels. Fields associated with ABPlace, ABCompany, or ABAttorney are irrelevant and ignored.

When an object is referenced, the server uses the datatype of the reference to understand the structure of the information in memory. For example, assume that there is an object named "anABContact" that stores information about an ABDoc. If there is a reference to this object with a datatype of "ABDoc", then the server knows that the object will have ABContact fields, ABPerson fields, ABPersonVendor fields, and ABDoc fields. If there is a reference to this object with a datatype of "ABPerson", then the server knows that the object will have ABContact fields and ABPerson fields, but it will assume that the object has no fields beyond the ABPerson fields. There may be additional fields with information at the ABPersonVendor and ABDoc levels, but the server will be unaware of them.

In some cases, a method or user interface container receives a reference to an object, and the reference uses a "high-level" datatype (such as ABContact). This reference cannot be used as-is to access fields below the specified level because the server assumes those fields don't exist. There is a programming technique known as casting in which you explicitly state the datatype for a subtyped object. This is done using the syntax "`as <datatype>`", where `<datatype>` is the datatype that you want to explicitly identify. When you cast a reference, you are providing the server with more information about the structure of the object. A casted reference can access fields that an uncasted reference cannot.

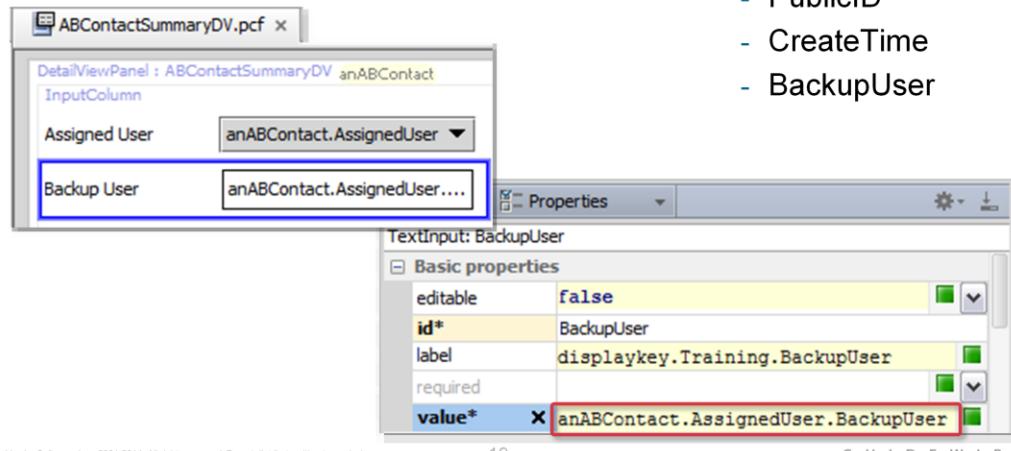
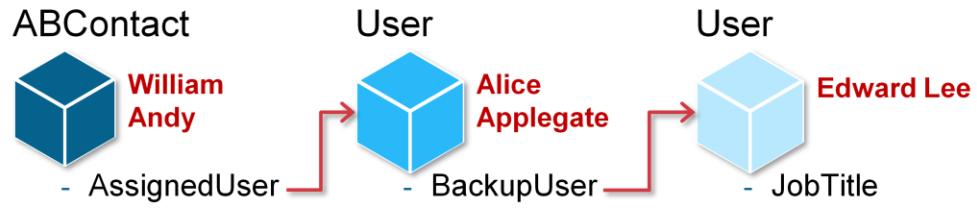
# Pointing to fields at subtype level

- Cast the object to subtype
  - root object has a supertype datatype and the desired field is at subtype level
- Syntax
  - `(object as Subtype).subtypeField`
  - `(anABContact as ABPerson).Gender`

The diagram illustrates the inheritance relationship between ABContact and ABPerson. ABContact is the supertype, represented by a yellow box containing 'EmailAddress'. ABPerson is the subtype, represented by a green box containing 'FirstName', 'LastName', and 'Gender'. An arrow points from ABPerson up to ABContact, indicating that ABPerson inherits from ABContact.

The screenshot shows the 'DetailViewPanel : ABContactDetailsPersonDV.pcf' interface. On the left, there's a panel titled 'Tax Info' with fields: 'Tax ID' (type '(anABContact as...)'), 'Tax Filing Status' (type '(anABContact as...)'), 'Date of Birth' (type '/.../...'), 'Gender' (type '(anABContact as...)', highlighted with a red box), and 'Marital Status' (type '(anABContact as...)'). On the right, the 'Properties' panel shows the 'Input: Gender' configuration. Under 'Basic properties', the 'value\*' field is set to `(anABContact as ABPerson).Gender`, which is also highlighted with a red box. The 'GUIDEWIRE' logo is visible in the bottom right corner of the interface.

# Pointing to fields on related objects



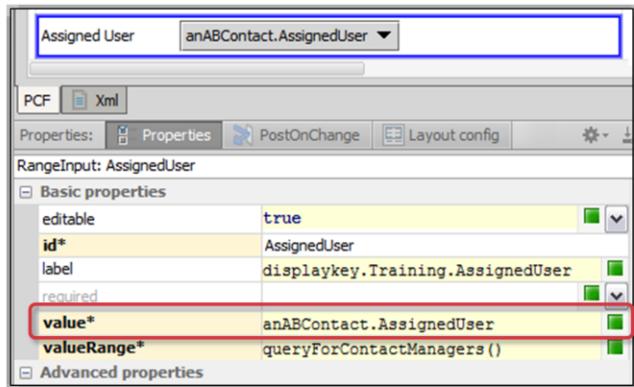
© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

19

G U I D E W I R E

In some cases, the desired field is on a related object. The syntax to reference fields on related objects is: `object.foreignKeyToRelatedObject.Field`.

## Step 3b: Specify the value property



- Value property defines the data field
- If the field is a "data" field, can set the editable property of the widget to "true"

- Specify field using dot notation
- Reference direct object field or related object field
- Only data fields are editable

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

20

G U I D E W I R E

For a given widget, the value property identifies the data field.

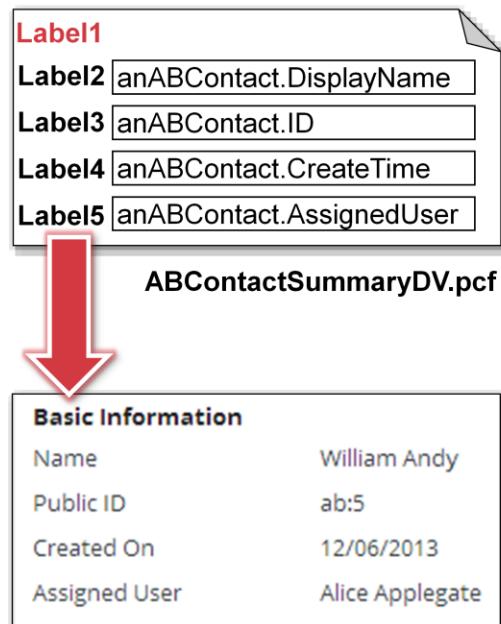


## Lesson outline

- Atomic widget fundamentals
- Creating atomic widgets
- Binding widgets to the data model
- **Widget labels and display keys**
- Optional widget properties
- Deploy the PCF and display keys

# Structure, function, and text

- Use display keys for label values
  - Easily localize text and accommodate users from various locales
  - Allows for labels with dynamic content
- Guidewire recommends PCF file should not contain "hard coded" display text such as label text
  - PCF file defines structure and function of containers



# Display key example

displaykey.Training.Name	Value
Locale	
English	Name
French	Nom
Japanese	名前
Spanish	Nombre



User: John Miller  
Locale: English (United States)

**Basic Information**

Name	William Andy
------	--------------



User: Jean-Luc Monet  
Locale: Français (France)

**L'information de Base**

Nom	William Andy
-----	--------------

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

ABContactSummaryDV.pcf

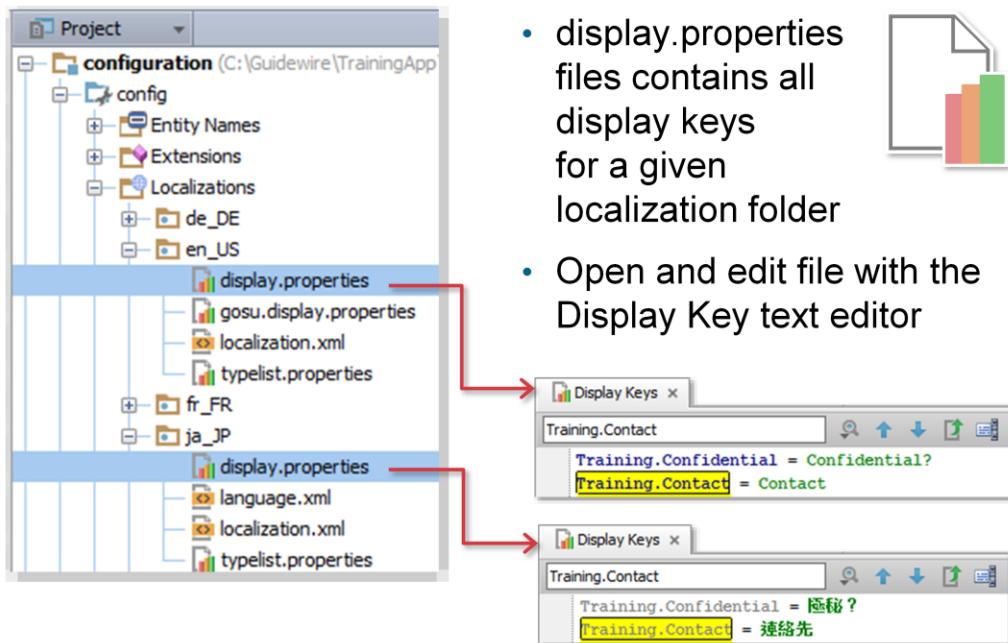
The screenshot shows the 'Properties' panel for a 'Detail View Panel' named 'ABContactSummaryDV'. Under the 'Input Column' section, there is a 'Basic Information' group. Within this group, there is an 'Input: Name' field. The 'label' property is set to 'displaykey.Training.Name', which is highlighted with a red box. The 'value\*' property is set to 'anABContact.DisplayName', also highlighted with a red box. Other properties shown include 'editable' (set to 'false'), 'id\*' (set to 'Name'), and 'required' (set to 'true').

23

G U I D E W I R E

A display key is a text string displayed on the user interface. Every display key has one or more localized values. When a PCF file references a display key, the application converts the display key to one of the localized values depending on the user's internationalization language settings. In the example above, the display key "Name" is converted to "Name" for users with their language set to English, "Nombre" for users with their language set to Spanish, and "Nom" for users with their language set to French.

# display.properties



- display.properties files contains all display keys for a given localization folder
- Open and edit file with the Display Key text editor

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

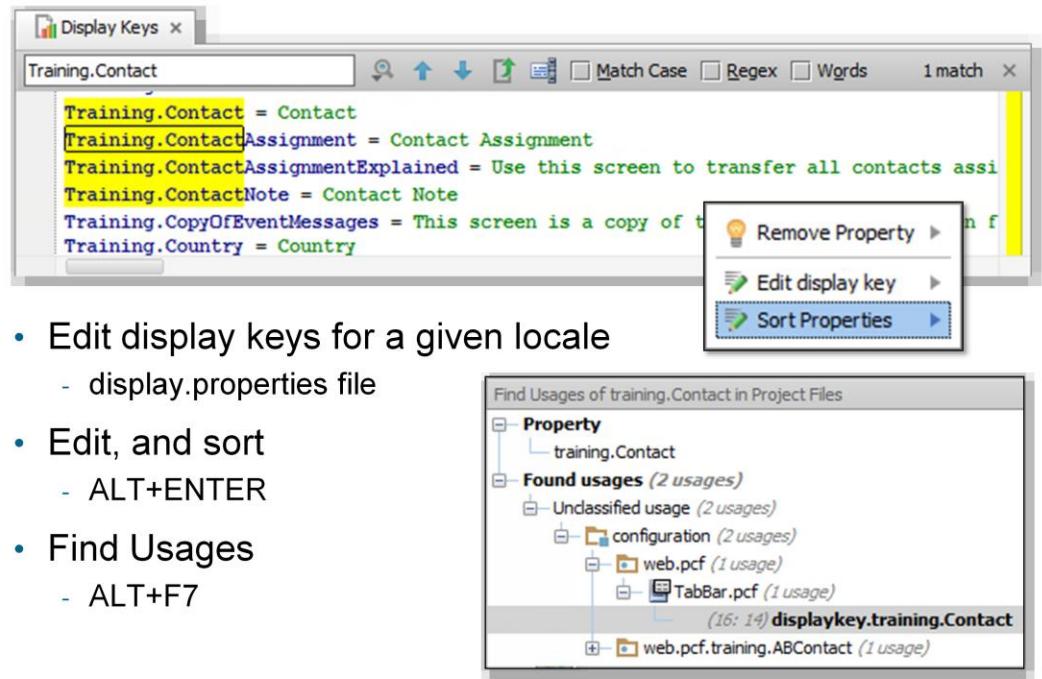
24

G U I D E W I R E

Because the display.properties file is associated with a physical folder that reads "locale", there is often confusion between the difference between language and locale. Locale settings relate to national and cultural settings for how to display information such as numbers, currency, and dates. Language settings relate to the user selected language which localizes the application in that selected language.

Guidewire applications that are configured to support additional languages use the related display.properties file and its defined display key values. If no display key is defined for the selected language, the application uses the default language display.properties file. A setting in the config.xml file defines the default language for a Guidewire application.

## Display Keys text editor



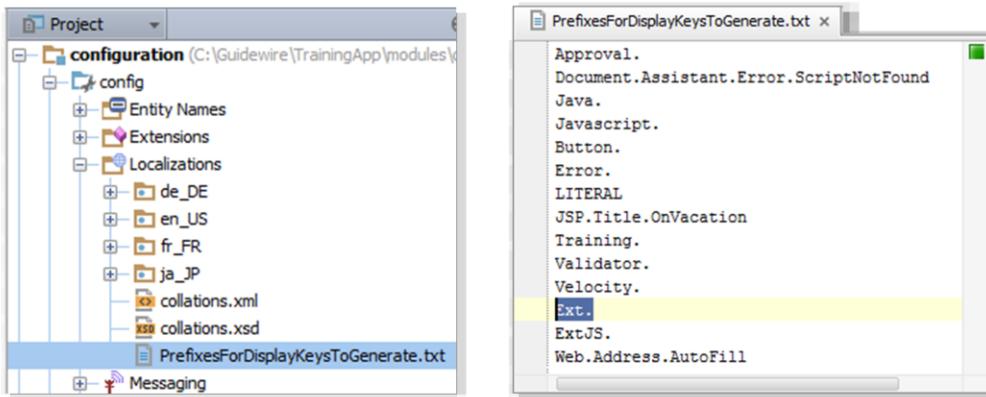
© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

25

G U I D E W I R E

Often when working with PCF files and widget properties, you want to use a display key for a localized value as in the case of defining a value for widget's label property. You can create, edit, and remove display keys with the Display Keys text editor.

# Display keys prefixes



- PrefixesForDisplayKeysToGenerate.txt
  - Contains prefixes for display keys
  - Not required, but helpful for organizing prefixes
- For customers, naming convention recommendation is "Ext"

## Step 4: Specify a display key for the label

The screenshot shows the Guidewire Studio interface. On the left, there's a properties panel for a 'TextInput' component named 'BackupUser'. Under the 'Basic properties' section, the 'label' property is set to 'displaykey.Training.BackupUser'. A red circle with the number '1' is placed over this field. Below it, the 'value\*' property has a tooltip 'Create Display Key' (red circle '2') pointing to a button. To the right, a 'Create Display Key' dialog box is open. It has a 'Name' field containing 'Training.BackupUser' (red circle '3'). Under 'Values', there's an 'en\_US' entry with the text 'Backup User'. At the bottom right of the dialog are 'OK' and 'Cancel' buttons. At the very bottom of the screen, there's a toolbar with icons for Help, Undo, Redo, and Refresh, with the Refresh icon (red circle '4') being checked.

1. Enter the displaykey name in the label property
2. Click Create Display Key
3. In the Create Display Key dialog, for the locale, enter the text value
4. Refresh the PCF

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

27

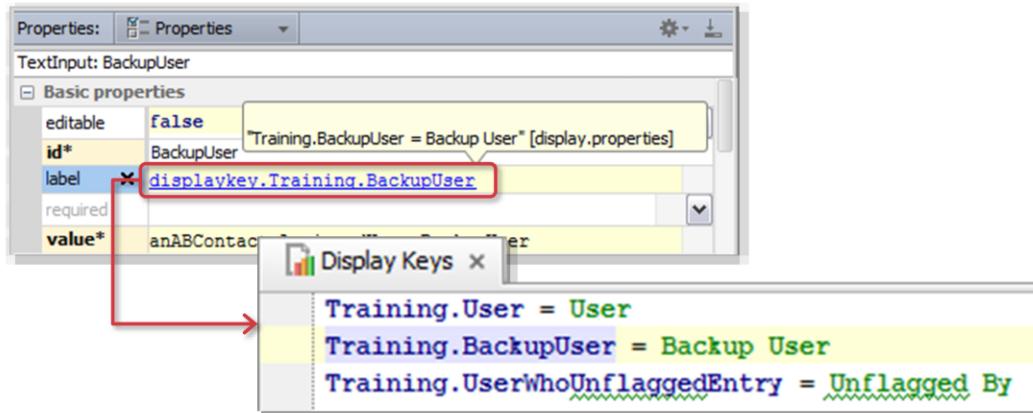
G U I D E W I R E

When you defined a value for a label property, you have the option to create a new a display key for the value or specify an existing display key for the property value.

1. Enter a display key name in the label property. A display key name cannot have a space in it. After you enter a new display key name, you will be prompted to create a display key.
2. Next you should enter whatever you want to appear as the label for the new widget. You can enter in English, Japanese, German, and French.
3. Save the additions you made in the display key editor (you can verify that the new key is in the display.properties file in the Localizations).
4. To see the changes in Studio, go back to Studio and click on the Refresh PCF button.

The slide example shows a display key prefix named "Training". All TrainingApp display keys have the Training prefix. The recommended naming convention for customer display keys, however, is to a standard prefix such as "Ext". To reference a display key, always start with displaykey followed by a prefix and then the name of the display key.

## Edit a display key label value



- **CTRL+HOVER**
  - Shows value of defined display key
- **CTRL+CLICK**
  - Navigate to display.properties

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

28

G U I D E W I R E

Both keystroke techniques work for a variety of items.

For example, a detail view panel can be embedded in a screen using a `PanelRef` widget. The `PanelRef` has a property that names which detail view panel to embed. From the `PanelRef` widget, you can **CTRL+CLICK** the name of the referenced detail view panel to open it in the PCF Editor.

# Display keys with dynamic content

The screenshot shows the Guidewire Studio interface with three main windows:

- Display Keys**: A dialog box showing three display keys:
  - Training.AddressType = Address Type
  - Training.AddressesCount = Addresses ({0})
  - Training.AddressesLDV = Addresses
- Properties**: A panel showing the properties for a **LocationRef** object. The **label** property is set to `displaykey.Training.AddressesCount(anABContact.AllAddresses.length)`.
- Contact Details**: A table showing contact information. The **Addresses** column has a value of "2".

A red box highlights the third display key in the **Display Keys** dialog. Red arrows point from the highlighted text in the **Display Keys** dialog to the **label** property in the **Properties** panel and to the value in the **Addresses** column of the contact details table.

A display key can have one or more arguments. Each argument is noted by "`{X}`", where X is an integer. If there is only one argument, then the integer should be 0. If there are multiple arguments, then the arguments should be numbered starting with 0. When there are multiple arguments, the first argument listed in the label property is assigned to the `{0}` position, the second to the `{1}` position, and so on. In the slide example, the label property specifies a display key with an argument. It is also passed a value, specifically the length of the `AllAddresses` array. The label is rendered using the named display key with the argument is the designated location.

As an example of a display key with multiple arguments, consider a display key "Training.DataChanged", which has the value: 'The object you are trying to update was changed by {1} at {0}. Please cancel and retry your change.'. At runtime, the values "Jan 13, 2009 11:32 AM" and "Alice Applegate" are passed to the display key. The message displayed would be: "The object you are trying to update was changed by Alice Applegate at Jan 13, 2009 11:32 AM. Please cancel and retry your change."

It is also possible for a display key to reference another display key. The syntax for this is: `{displayKeyToEmbed}`. An example is `Web.Admin.NewReportGroup{Web.Admin.AddReportGroup}`.

Embedding display keys can make UI changes both easier and more difficult. It can ease the process of changing a given term. For example, changing "POs" to "Purchase Orders" could be easier if the term exists in only one display key that is embedded in other display keys. It can complicate the translation process, however, if you want to implement a multi-lingual instance of the Guidewire application. Word order varies from language to language, and when you embed display keys in display keys, you start to "hard code" the word order. A given phrase may consist of contiguous words in one language but not in the other.

- `{#}` is a token for an input parameter data starting at a zero index
  - `{0}` is first parameter



## Lesson outline

- Atomic widget basics
- Creating widgets
- Binding widgets to the data model
- Widget labels and display keys
- **Optional widget properties**
- Deploy PCFs and display keys

## Examples of other optional properties

- inputConversion

Stock Symbol	Acme
--------------	------

↓  
ACME

- outputConversion align

5400-3256-2211-5423

↓

Credit Card	XXXX-XXXX-XXXX-5423
-------------	---------------------

- align

Total Score
68
96
87

- labelAbove

Employees		
Name	Job Title	Email Address
Eric Andy	Associate	
William Andy	Associate	

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

31

G U I D E W I R E

outputConversion is a Gosu expression property that modifies the value stored in the database before it is displayed in the user interface. For example, it can be used to mask all but the last 4 digits of a credit card number.

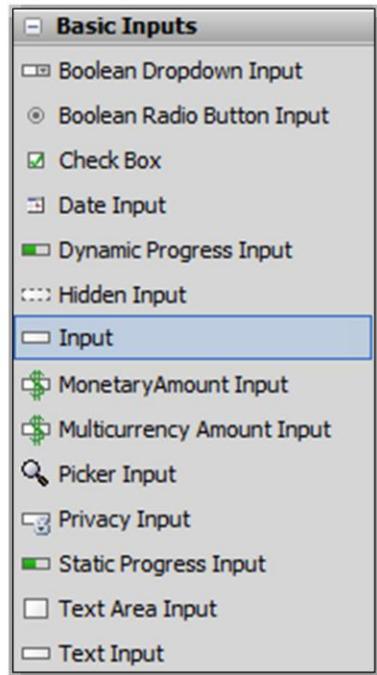
inputConversion is a Gosu expression property that modifies the value displayed in the user interface before it is stored in the database. For example, it can be used to convert any entered stock symbol into all upper-case characters.

align is a property that can be set to either "left", "center", or "right". It impacts the alignment of data in the corresponding widget.

labelAbove is a boolean property for widgets with labels and fields. When set to true, the widget's label is displayed above the field, rather than to the left of the field. In the screenshot above, the Occupation and Employer widgets have labelAbove set to true. The "Employment Info" widget is a label widget with no field.

# Input widget

- Input widget has only standard input properties
- Renders various data types:
  - Text field when underlying field is numeric or varchar
  - Date formatted field when underlying field is datetime
  - Dropdown when underlying field is typekey
  - Yes and no radio buttons when underlying field is boolean



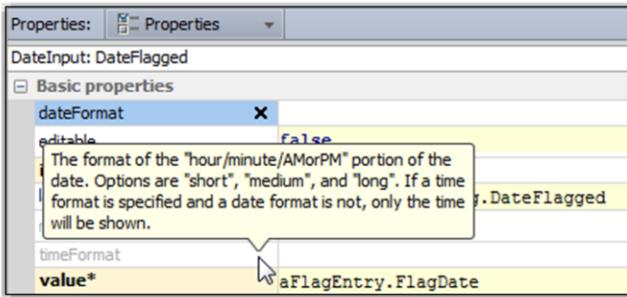
© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

32

G U I D E W I R E

It may seem negligible, but to resolve the underlying data type for the widget value takes extra application processing. In aggregate, widget resolution can be a performance issue. Wherever possible, use the input widget that best matches the needs of the user, widget special properties, and the underlying data type value.

# Property tooltips



- If you mouse over a property name, a tooltip displays Description text of that property from the PCF reference
- Does not display property value type or defaults

## Step 4: Optional widget properties (1)

The screenshot illustrates the configuration of a 'Status' field in a PCF file. The left side shows a user interface with a 'Status' field containing 'Unverified'. The right side shows the PCF configuration for this field, specifically the 'Basic properties' section where the 'editable' property is set to 'false'.

Property	Value
id*	Status
label	displaykey.Training.Status
required	
value*	aVendorEvaluation.Status
editable	false

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

34

G U I D E W I R E

The required variable for the VendorEvaluationDV.pcf file is aVendorEvaluation of the type VendorEvaluation. The VendorEvaluation entity describes the Status field. The Status field is a typekey defined by a value from the VendorEvaluationStatus typelist. For this entity typekey, the default value is unverified. The Status field can also contain a null value.

However, in the VendorEvaluationDV, the Status widget does not allow the Status property to be editable. This results in the value to always use the default value as described by the VendorEvaluation entity.

## Step 4: Optional widget properties (2)

The screenshot displays two windows side-by-side. On the left is a 'Vendor Evaluation' dialog box with three fields: 'Evaluator' (with an asterisk), 'Evaluation Date' (with a calendar icon), and 'Status' (set to 'Unverified'). A red box highlights the 'Evaluator' field. On the right is the 'VendorEvaluationDV.pcf' configuration window. It shows a 'DetailViewPanel' with an 'InputColumn' containing the 'Evaluator' field. Below this, the 'Properties' panel is open for the 'Evaluator' field, specifically the 'Basic properties' section. A red box highlights the 'editable' and 'required' properties, both of which are set to 'true'. The 'label' property is set to 'displaykey.Training.Evaluator' and the 'value\*' property is set to 'aVendorEvaluation.Evaluator'.

- By default, input widgets are visible, not required, and not editable
  - Editable – user can edit value
  - Required – value must be defined; asterisk (\*) denotes required field

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

35

GUIDEWIRE

The required variable for the VendorEvaluationDV.pcf file is aVendorEvaluation of the type VendorEvaluation. The VendorEvaluation entity describes the Evaluator field. The Evaluator field is a varchar data field that can be null. No default value is defined for the field.

However, in the VendorEvaluationDV, the Evaluator widget does not allow the Evaluator field to have a null value. The Evaluator widget defines properties for making the widget both editable and required. Required and editable fields show an asterisk in the user interface when the PCF is in edit mode.

If the VendorEvaluation entity defines the Evaluator field as non-null (null=false), then the Evaluator widget shows the field to be required in the case when the required property is undefined. If the required property is set to false, then the widget will override this behavior. The result could be a DBNullConstraintException when a user attempts to commit the data.

Visible is typically not set to false. It is either set to true or it is set to an expression that renders the widget visible or not visible based on conditional logic. Setting widget properties to conditional expressions is covered in the "Partial Page Update" lesson.

# Specialized widgets have extra properties

Properties:  Properties	
DateInput: DateFlagged	
Basic properties	
dateFormat	x long
editable	false
id*	DateFlagged
label	displaykey.Training.DateFlagged
required	
timeFormat	
value*	aFlagEntry.FlagDate

- Specialized widgets include additional properties for finer level of control

**Flag Entry** [Return to Summary](#)

Date Flagged	02/11/2014
<b>Flag Entry</b> <a href="#">Return to Summary</a>	
Date Flagged	Tue, Feb 11, 2014



© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

36

G U I D E W I R E

# Other common atomic input widgets

- Basic
  - Boolean Dropdown Input
  - Boolean Radio Button Input
  - Check Box Input
  - Date Input
  - Monetary Amount Input
  - Multicurrency Amount Input
  - Text Area Input
  - Text Input
- Range
  - Range Input
  - Range Radio Button Input
  - TypeKey Input
  - TypeKey Radio Button Input

# Data values and input widget examples

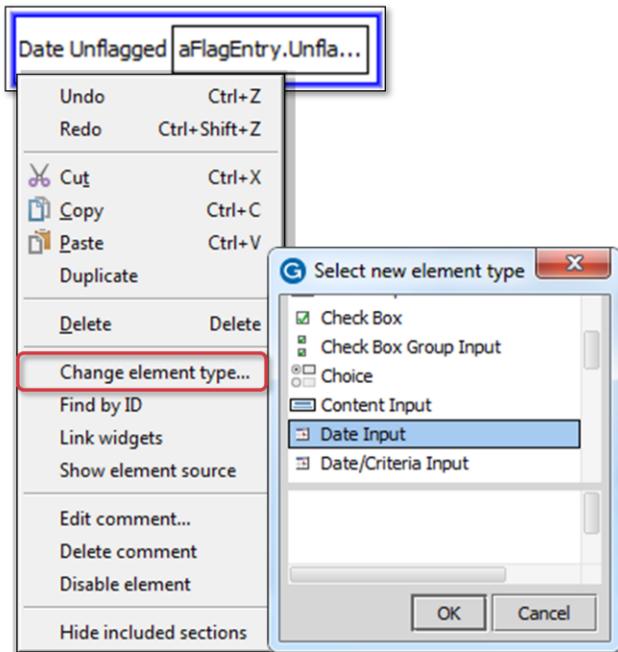
Data	Input Widget
varchar integer decimal	Text Input
varchar shorttext mediumtext longtext text	Text Area Input
date datetime	Date Input
money	Monetary Amount Input
bit	Boolean Dropdown Input Boolean Radio Button Input Check Box Input
typekey	TypeKey Input TypeKey Radio Button Input

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

38

G U I D E W I R E

# Change element type for atomic widgets



- Select widget element in PCF Editor canvas
- Right click to open context menu
- Change element type...
  - Opens dialog of possible substitutes
- Make change selection in dialog

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

39

G U I D E W I R E

"Change element type..." context menu command substitutes a different element for the selected element. The dialog contains a list of element types that you can substitute for the selected element within the constraints of the PCF schema.

# Documentation for all widget properties

This file is in: <Application>\modules\pcf.html

## Guidewire PCF Format Reference

### Input

[top](#)

A generic Input, which can be bound to any value. The type of the value widget (e.g., textbox, dropdown or checkbox) will be determined dynamically based on the value type, for example:

- Boolean - BooleanRadioInput
- Date - DateInput
- TypeKey - TypeKeyInput

If no such mapping is found, the default is to use a TextInput. In general, this tag should be used where possible in preference to a more specific tag, unless the special attributes of the specific tag are needed.

Child elements: [Reflect](#) [AbstractMenuItem](#)

Attribute:	Type:	Default:	Description:
action	statement		An optional action to take when the user clicks the value of this widget when the value is not being edited. See <a href="#">ActionBase</a> for list of available action prefixes. When specified, the value of this widget will be rendered as a link when the value is not being edited.
id	String		Reference ID of the widget. This does not have to be globally unique; it only has to be unique among the widget's siblings
visible	Expression (Boolean)		If used instead of "editable", specifies whether the Input is editable when data is being created

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

40

G U I D E W I R E

The PCF Format Reference defines the properties for every widget, including the type of value the property takes and a description of the widget. Note that the PCF Format Reference refers to widget properties as "Attributes".

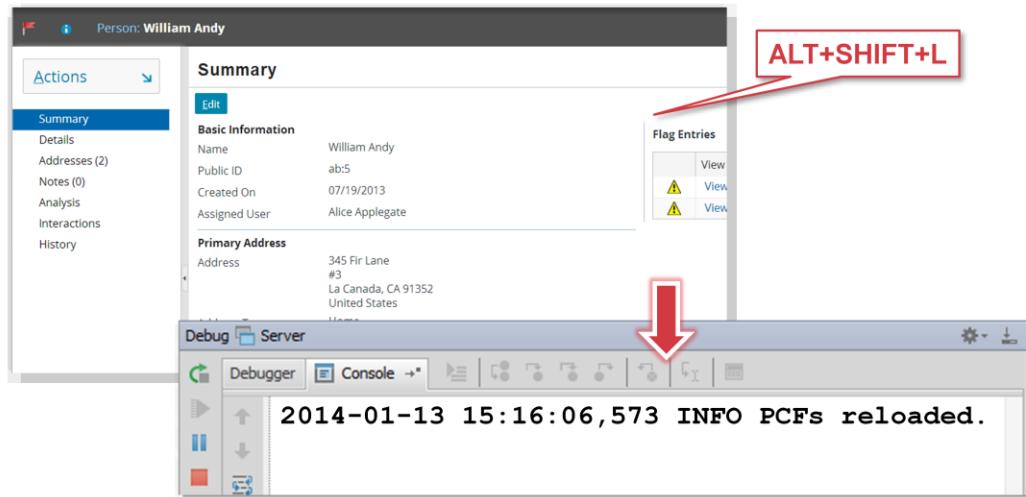
The PCF Format Reference is located in <ApplicationRootDirectory>\modules\pcf.html.

## Lesson outline

- Atomic widget basics
- Creating widgets
- Binding widgets to the data model
- Widget labels and display keys
- Optional widget properties
- Deploy PCFs and display keys

# Internal debug tools: Reload PCFs

**ALT+SHIFT+L**



- Reloads all Page Configuration Files
- Command window or Guidewire Studio Console window (Debug / Run) details output

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

42

G U I D E W I R E

If you are running an open application project in Guidewire Studio and if internal tools are enabled, you can reload all the page configuration files and displaykeys for the server.

If you reload PCF files while in edit mode, you may experience unpredictable results. For the current location, where there is a data modification in progress, the new PCFs may not be reloaded. Therefore, Guidewire recommends reloading PCF files while in read-only mode as it provides for more predictable results.

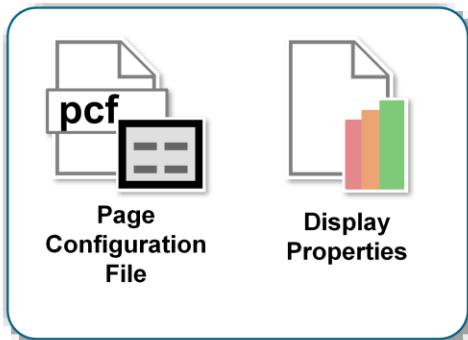
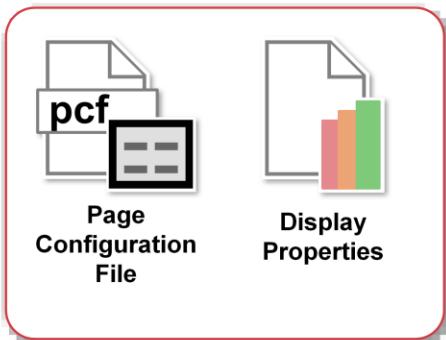
## Step 6: Deploy PCF files and display keys

Restart Server

- PCFs read at server startup

Reload PCFs

- ALT+SHIFT+L
  - Internal debug tools enabled
- Internal Tools
  - Reload → Reload PCF Files



© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

43

G U I D E W I R E

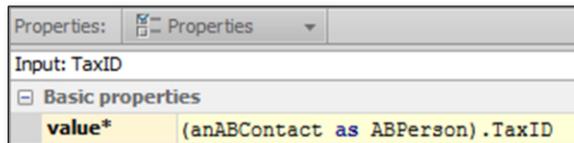
It is also possible to reload PCF files using the Guidewire API and/or internal server tools. The Reload PCF command can be found on the Reload page in Internal Tools. To access Internal Tools, you must log in as an administrator user, e.g., su/gw. Then, use ALT+SHIFT+T. In the tab bar, select Internal Tools --> Reload. On the Reload page, click the Reload PCF Files button. The Reload PCF Files button calls the static method `gw.api.tools.InternalToolsUtil.reloadPCFs()`.

## Lesson objectives review

- You should now be able to:
  - Create atomic widgets
  - Bind an atomic widget to the data model
  - Create and modify widget labels
  - Use optional widget properties

## Review questions

1. What must you specify in a widget's "value" property?
2. When would you see the keyword "as" in the value property?
3. What is a display key?
4. If you create a widget and specify only the ID, name, and label, is the widget visible? Editable? Required?
5. The screenshot shows a standard input widget. The customer requires a "long" date format.  
How could you implement this requirement?



Flag Entry <a href="#">Return to Summary</a>	
Date Flagged	02/11/2014

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

45

G U I D E W I R E

### Answers

- 1) You must specify the field in the data model that the widget is bound to.
- 2) You would see the "as" when the container's base object is subtyped, and the field to which the widget must be bound is at one of the subtype levels.
- 3) A display key is an abstract display value referenced by a widget's label property. It can have one or more locale-specific values, and when the UI is rendered, the value matching the user's locale is used.
- 4) The widget will be visible, not editable, and not required.
- 5) You would need to recreate the widget using a date input. The date input widget contains properties to more finely control the widget behavior, such as the date format.

# Notices

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

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

All other trademarks are the property of their respective owners.

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

This file and the contents herein are the property of Guidewire Software, Inc. Use of this course material is restricted to students officially registered in this specific Guidewire-instructed course, or for other use expressly authorized by Guidewire. Replication or distribution of this course material in electronic, paper, or other format is prohibited without express permission from Guidewire.

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

© Guidewire Software, Inc. 2001-2014. All rights reserved. Do not distribute without permission.

46

G U I D E W I R E