
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

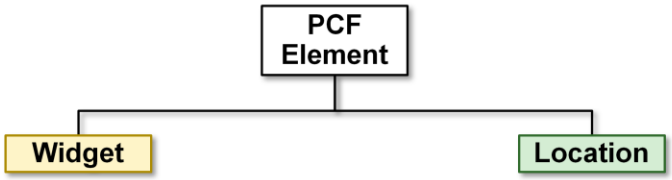
- By the end of this lesson, you should be able to:
  - Describe the user interface architecture for Guidewire applications
  - Describe the kinds of Page Configuration Files (PCF)
  - Create new PCF folders and files
  - Open and edit PCF files
  - Deploy PCF files

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.

# Lesson outline

- User interface architecture
- PCF files
- Create and open PCF files
- Modify PCF files
- Deploy PCF files

# Guidewire application UI framework




**Displayable elements of the user interface rendered into HTML**

**Navigable places in application that a user or the application itself can navigate to**

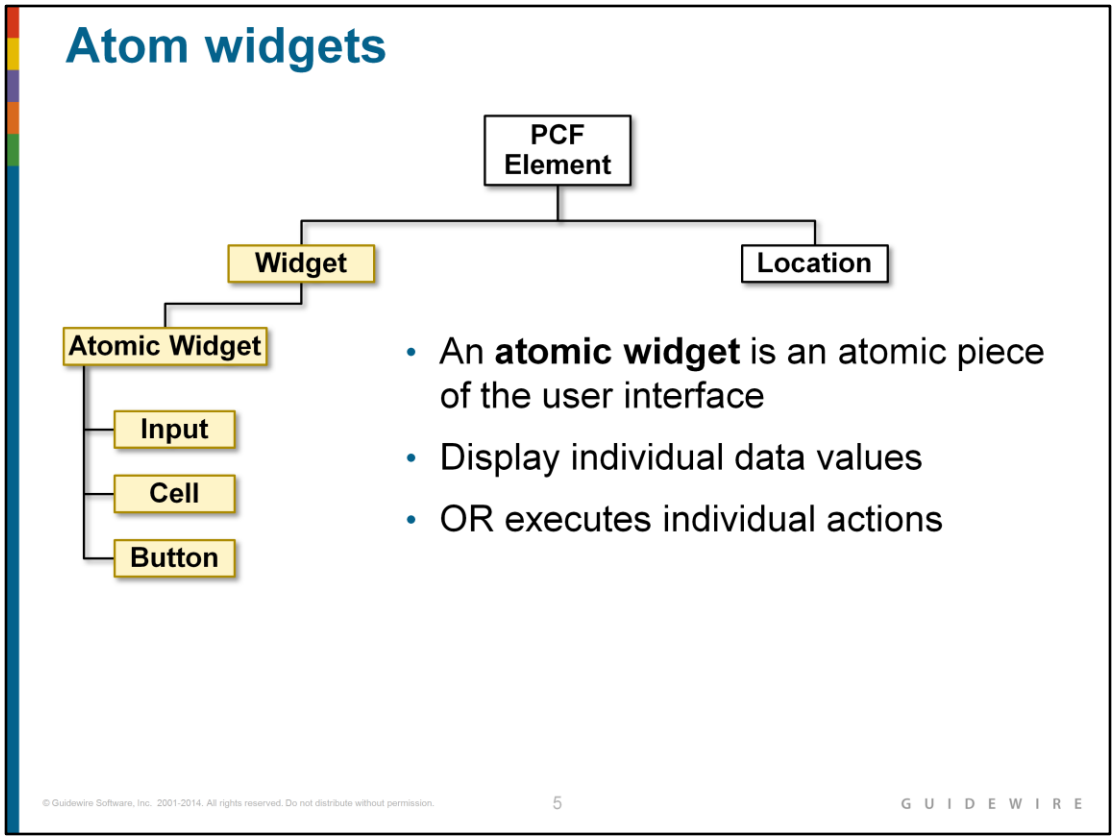
- The **PCF (Page Configuration File)** object model is a proprietary application framework used to create all Guidewire end-user interfaces
- Widget and Location are high-level categories for PCF elements




A widget is a PCF element that can be converted into HTML and displayed.

A location is a PCF element that a user (or the application itself) can navigate to. It is used to define how users move from one area of user interface to the next.

Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements.



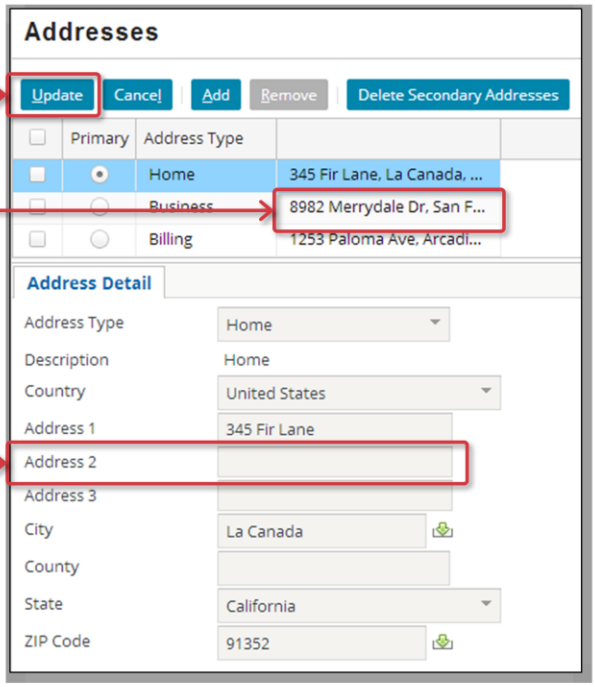

Atomic widgets are individual field items such as inputs, cells, or buttons. They will always be defined within a container widget or location.

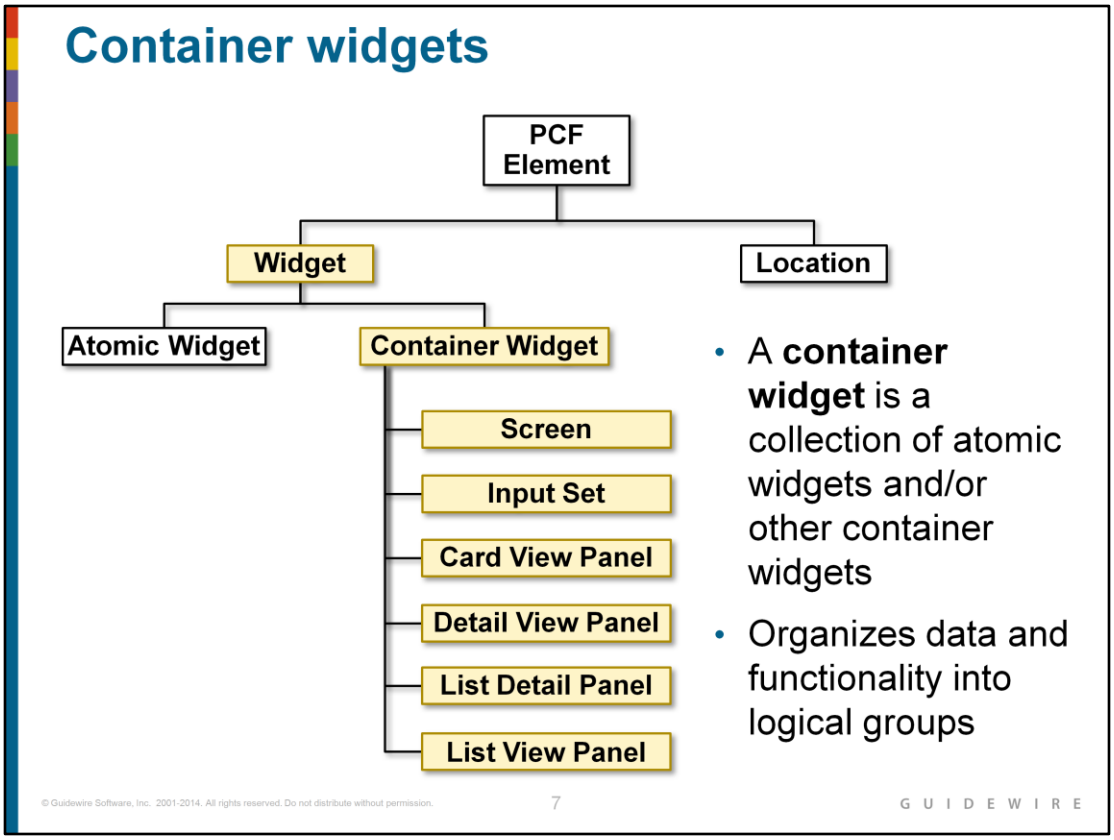
Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Atomic Widget is also a conceptual representation. There is no <Atomic Widget /> element.

The PCF object model is container-based. Each screen element is modeled as an object, which may contain other objects. The hierarchical structure simplifies the task of locating and modifying visual elements. Furthermore, each element can be declared as an independent and therefore reusable element.

# Atomic widgets: examples

- Button
- Cell
- Input





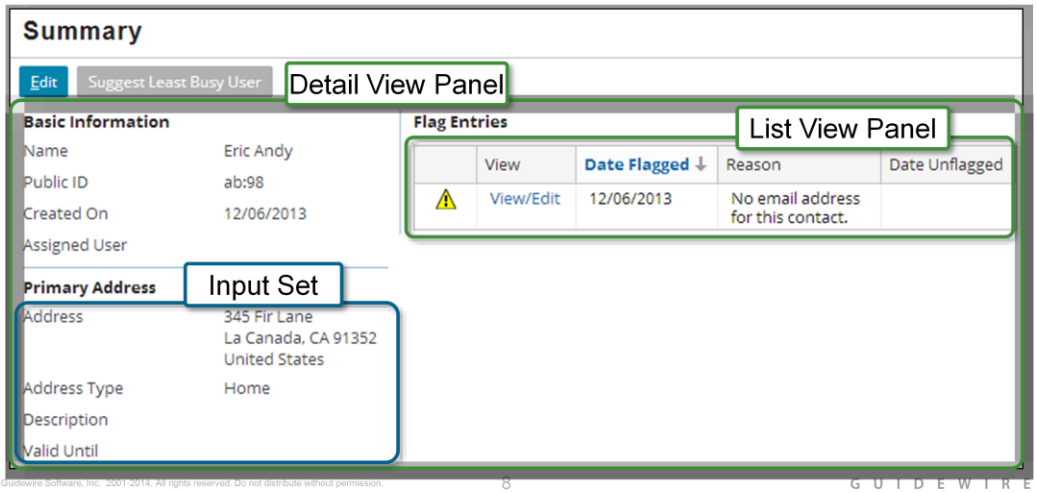

Container widgets hold other widgets. Each one can be defined either in its own file or as a child container within some other PCF element file.

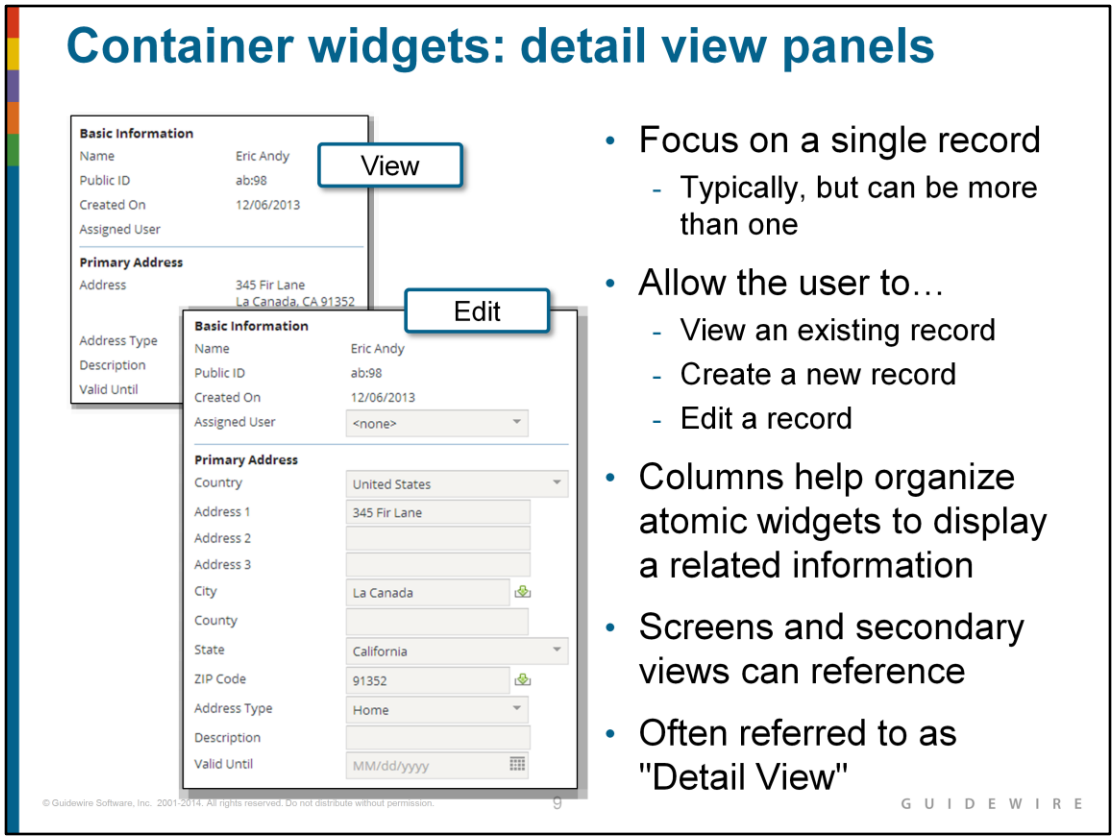
Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Similarly, both Atomic Widget and Container Widget are conceptual representations. There are no <Atomic Widget /> or <Container Widget /> elements.

The PCF object model is container-based. Each screen element is modeled as an object, which may contain other objects. The hierarchical structure simplifies the task of locating and modifying visual elements. Furthermore, each element can be declared as an independent and therefore reusable element.

# Primary views

- A primary view is a reusable **panel** such as a detail view panel or list view panel that organizes atomic widgets.
- An **Input Set** is a reusable collection of atomic widgets for detail view panels



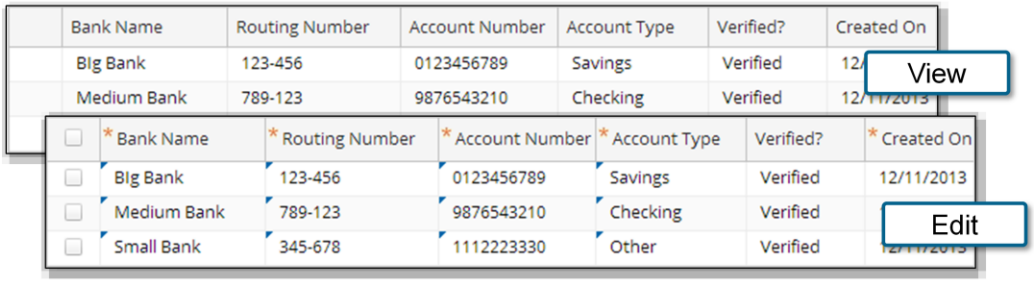



Detail view panels are generally designed to use widgets to organize and manage a collection of related pieces of information that constitute a single record. The most obvious example is the set of individual fields of an instance of an entity such as User, Claim, or Policy. There are no constraints imposed by the system as to what information can be displayed by a detail view, however; the view could present information from multiple entities that do not even have to be related to one another, in addition to information that does not derive from an entity at all (such as calculated or static values).

The purpose of a detail view panel, however, and the recommended practice behind their use, is to present the user with all of the detailed information that is relevant to that interface, and no information that is irrelevant. In general this means displaying information about the fields of a particular entity and possibly fields from a small number of closely related entities.

Detail view panels must contain at least one column and can contain as many columns as necessary to present the widgets in the most effective way for the user.

# Container widgets: list view panels



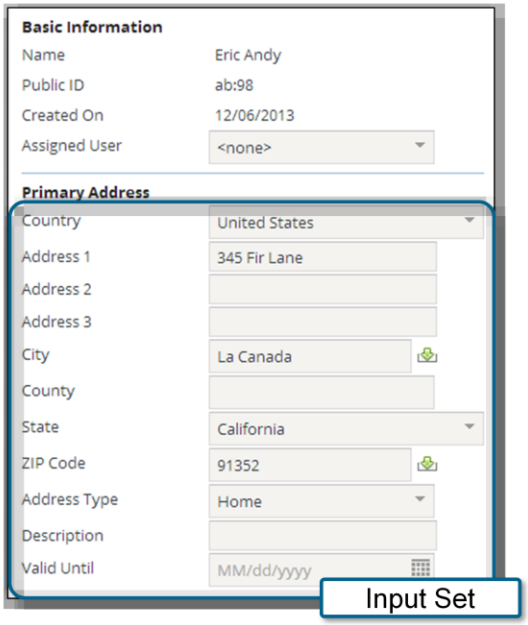

- Table organizes information that allows the user to view information about multiple records at the same time
- Uses a small number of atomic widgets to display the most relevant information
- Screens and secondary views can reference
- Often referred to as "List View"





Just as a detail view panel is designed to allow the user to view detailed information about a single record, a list view panel is designed to allow the user to view summary information about a collection of records.

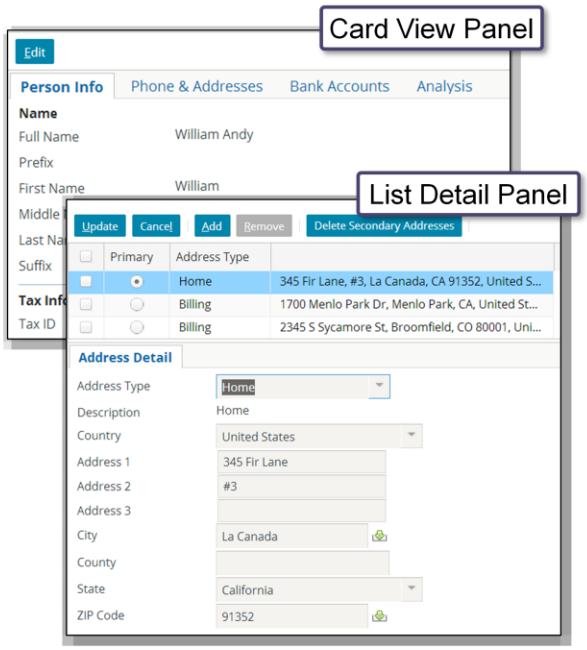
# Container widgets: Input Sets




- Can contain atomic widgets
- Can be reused by detail view panels
- Cannot be referenced by secondary views
- Cannot have a toolbar directly associated with them

# Secondary views

- A **secondary view** organizes primary views
- A **Card View Panel** is a collection of cards, with each card containing detail view panels or list view panels
- A **List Detail Panel** contains a top list view panel and a bottom view panel that display data about the selected list item

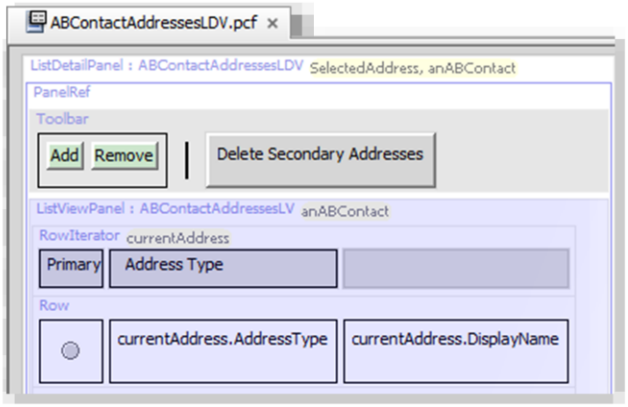


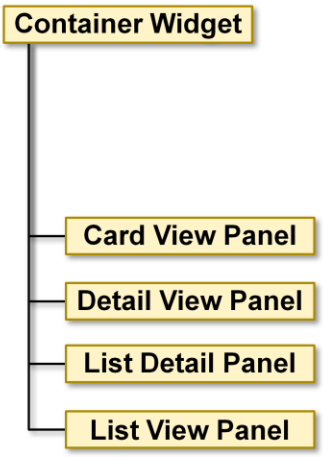




A Card View Panel is often referred to as a Card View. A List Detail Panel is often referred to as a List-Detail View.

# Panels




- A **panel** is a container which has a structured UI layout, shares a common placement on a PCF page, and can be rendered with a Toolbar and/or a Title bar that controls the panel behavior
  - Toolbar and Title bar require a containing PanelRef





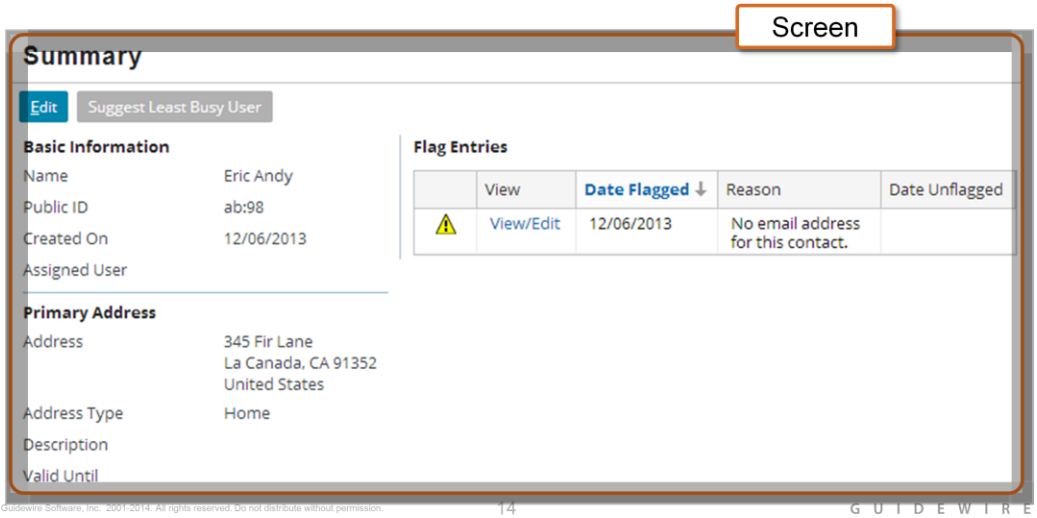
All panels share the common placement on a PCF page, for example, when defined directly on a Screen, Card, or Panel Set.

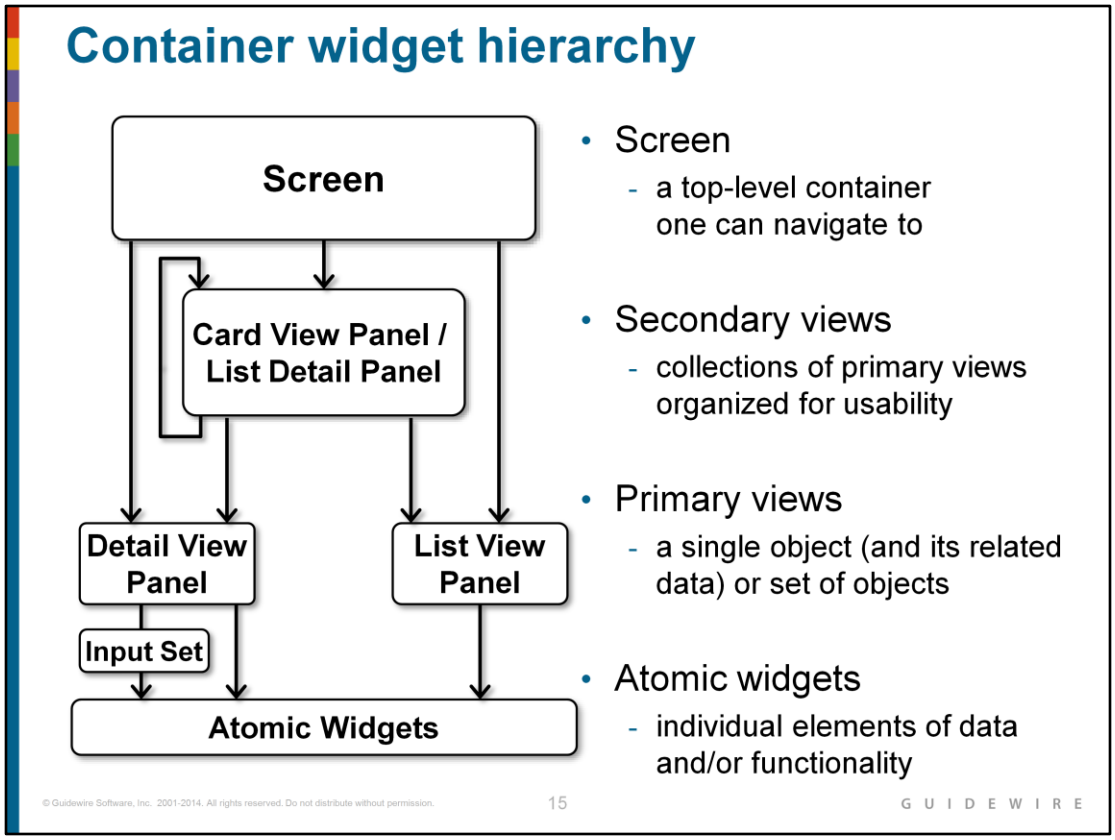
A panel and can be rendered with an associated Toolbar and/or Title Bar. Edit Buttons are typically associated with a Toolbar defined in a Screen.

Often, panels are placed within a Screen using a PanelRef widget. Although not a defining characteristic of panels, panel placement using a Panel Ref helps illustrate the distinction between primary views and secondary views.

# Container widgets: screens

- Screens are top-level containers
- Every atomic widget, primary view, and secondary view is contained (directly or indirectly) in a screen

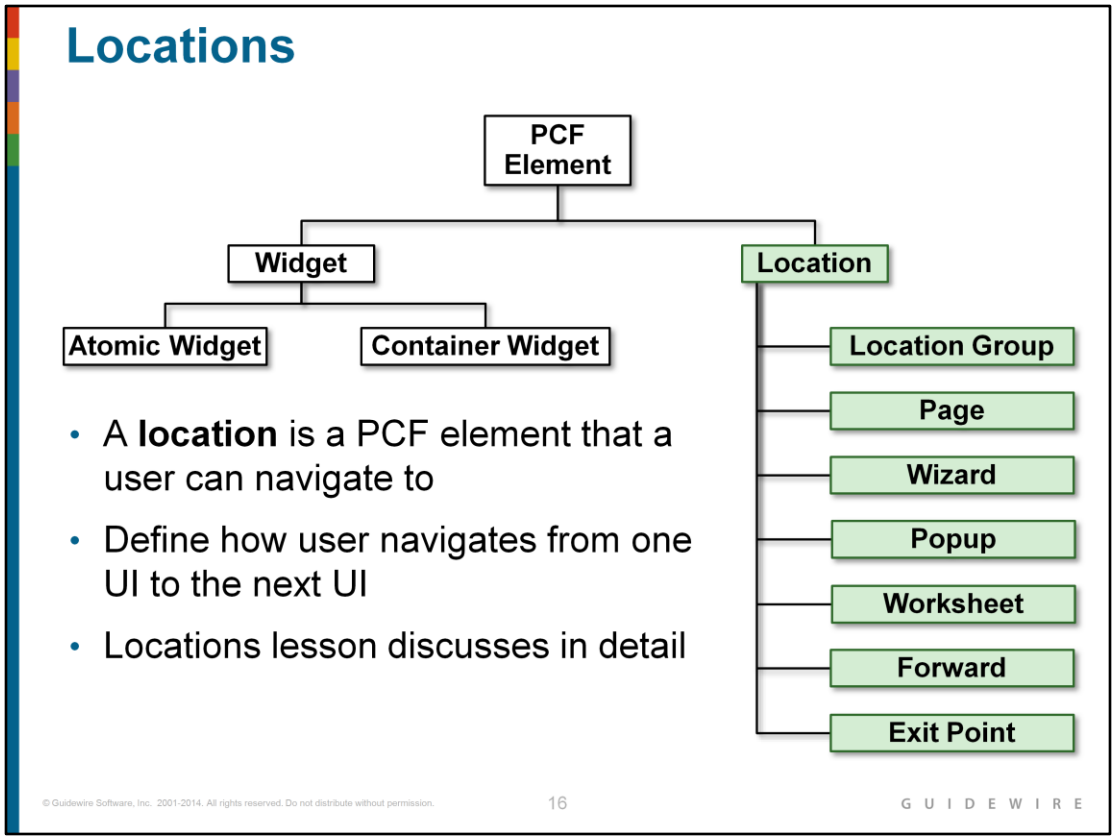




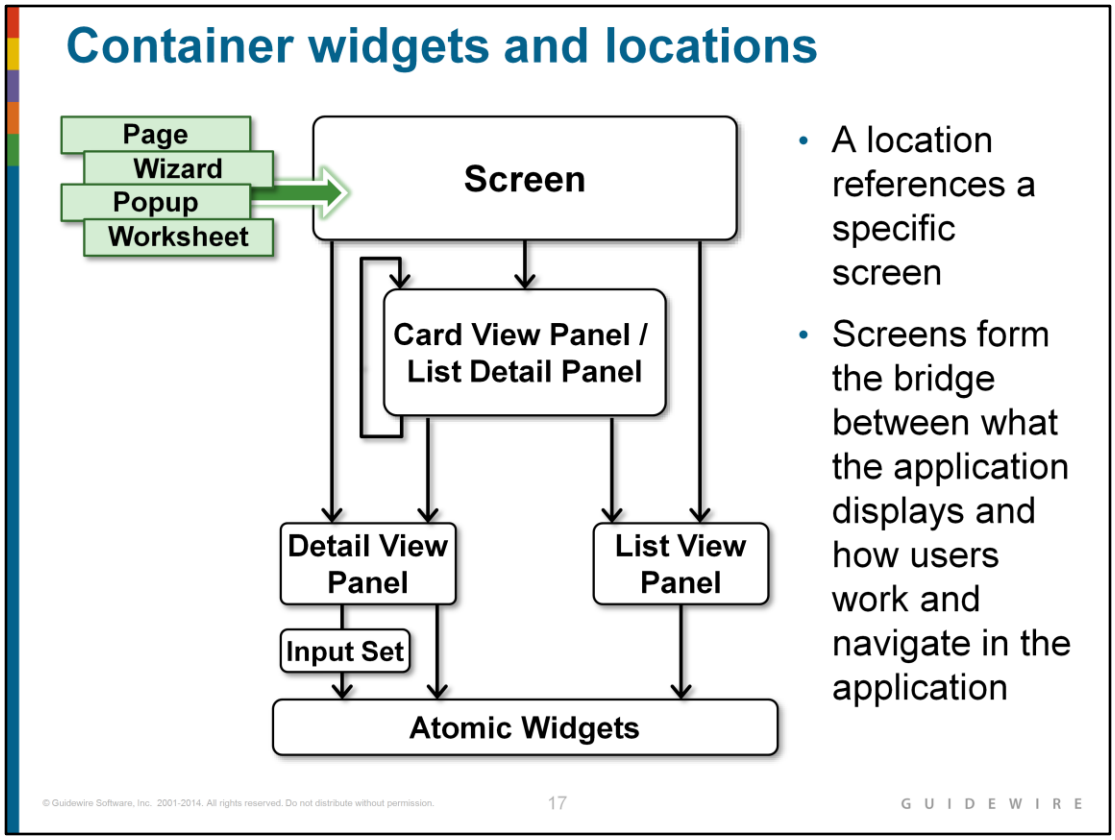

Container widgets can be classified in four categories:

- The lowest level is comprised of atomic widgets.
- Atomic widgets can be directly contained only by primary views (detail view panels and list view panels). The purpose of a primary view is to organize atomic widgets into logical groups.
- Secondary views (card view panels and list detail panels) organize primary views. Secondary views cannot directly contain most atomic widgets.
- Screens are the top-level containers. Screens are used to connect what is displayed in the user interface with how users navigate through the user interface. A screen can directly contain both primary views and secondary views, but it cannot directly contain most atomic widgets.

Input sets do not readily fall into any of these categories. Input sets fit some of the attributes of primary views. They can contain atomic widgets and organize them into logical groups. Input sets cannot be referenced directly by secondary views or screens. Input set must be embedded into detail view panels and cannot have toolbars directly associated with them.

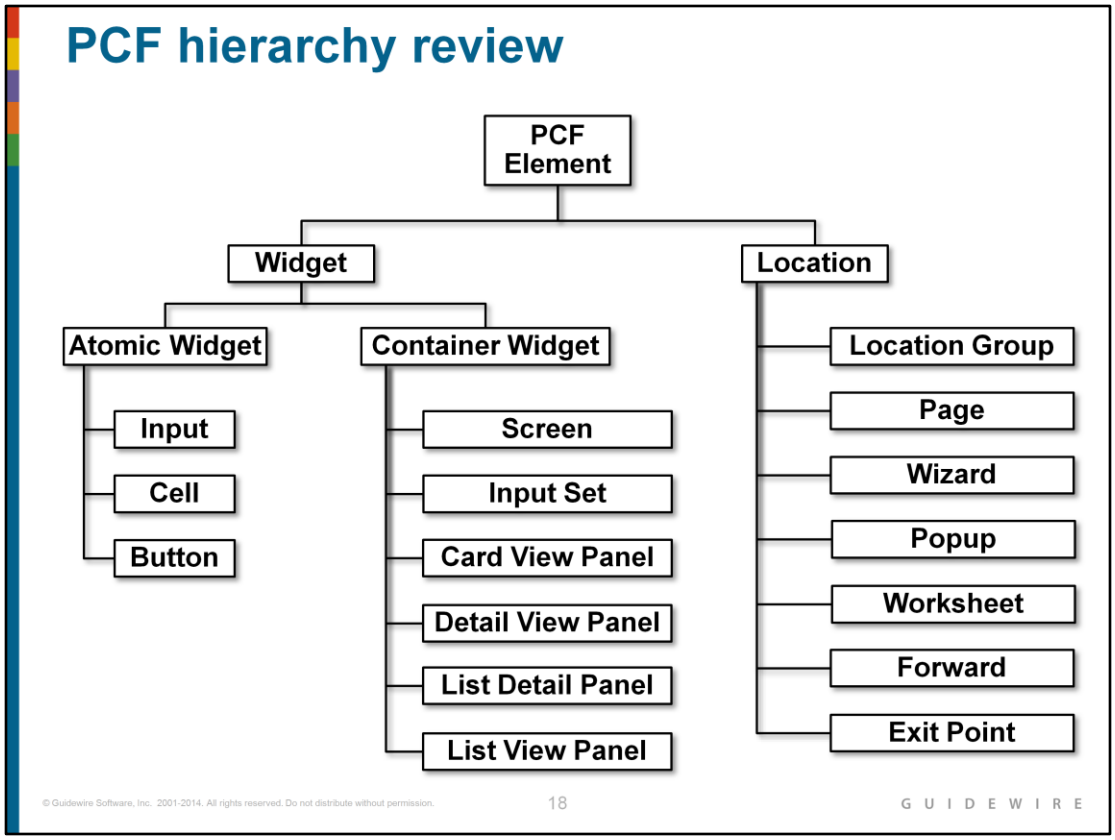



Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements.




Notice that only a subset of locations reference screens:

- LocationGroups reference a collection of Page locations, which then reference screens
- Forwards redirect to any other location types, which may contain screens
- Exit Points reference external URLs, which define non-application screens



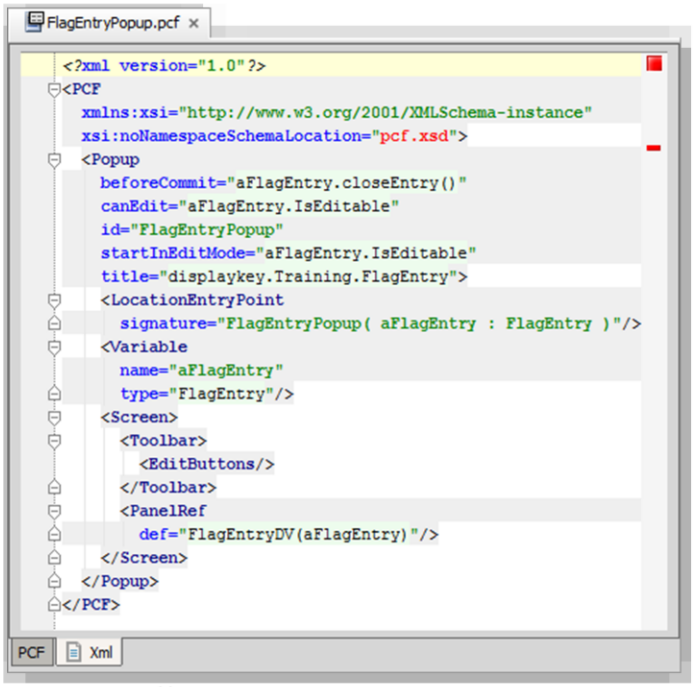

At the top level, PCF elements are split between widgets and locations. This lesson only addresses the high-level classifications of widgets (both atomic and container) and locations. Discussions of specific widgets and locations will follow in future lessons.

# Lesson outline

- User interface architecture
- PCF files
- Create and open PCF files
- Modify PCF files
- Deploy PCF files

# Page Configuration File (PCF)

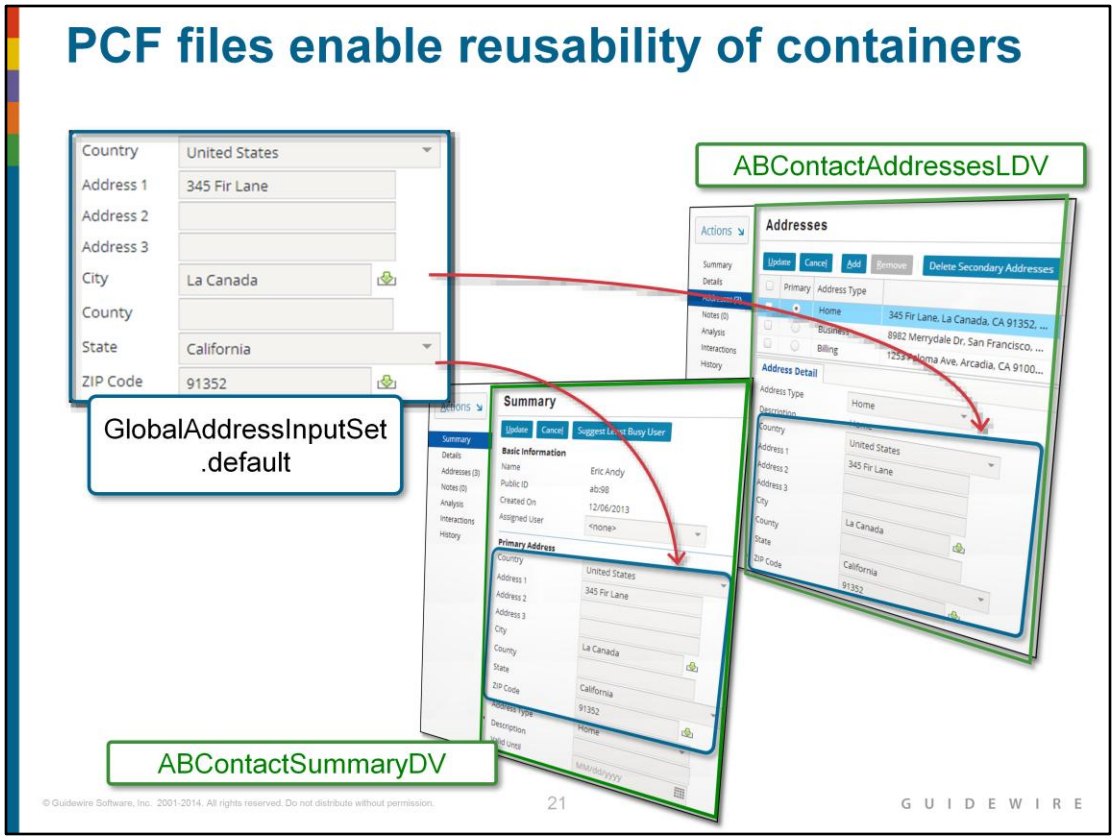
- A **PCF file** is an XML file that defines a location or container widget
- May also:
  - Define child containers
  - Reference other containers
  - Include atomic widgets



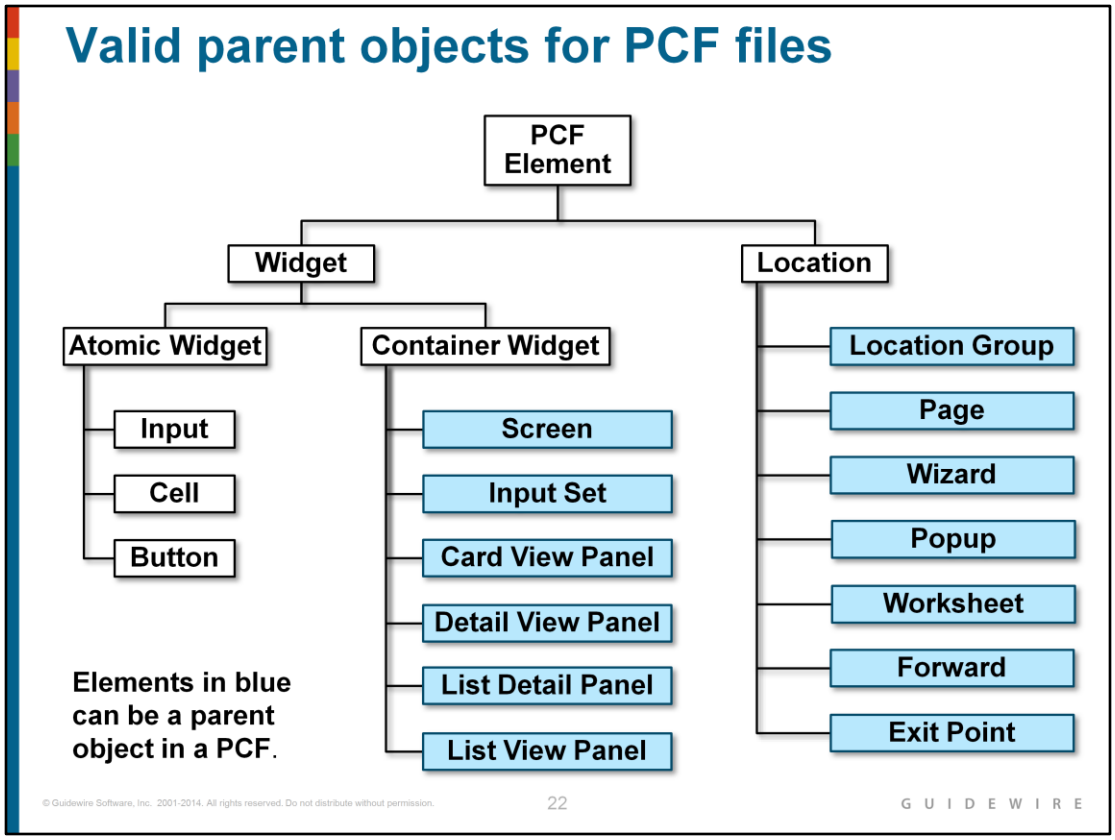




A PCF is an XSD-validated XML document containing elements that describe the structure, layout, and behavior of the web user interface. The majority of this content is defined at the application level, though some is defined at the platform level.

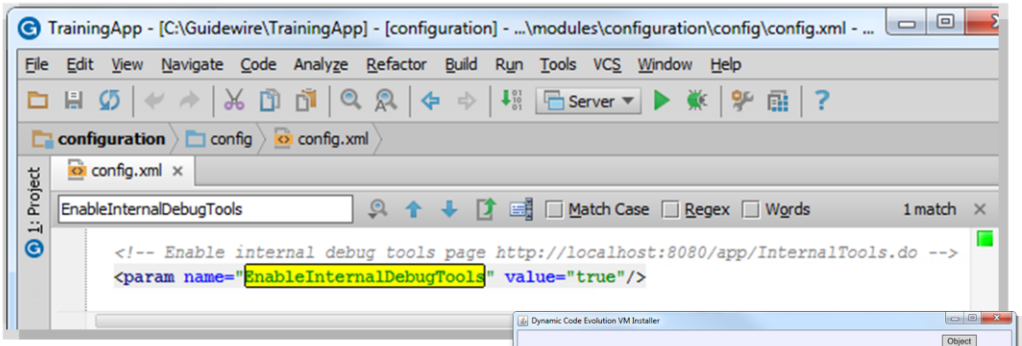



GlobalAddressInputSet is a globalization input set and is modal. Modal widgets are discussed in greater detail later in this course. It is not a requirement for a PCF to be modal in order for it to be reused.

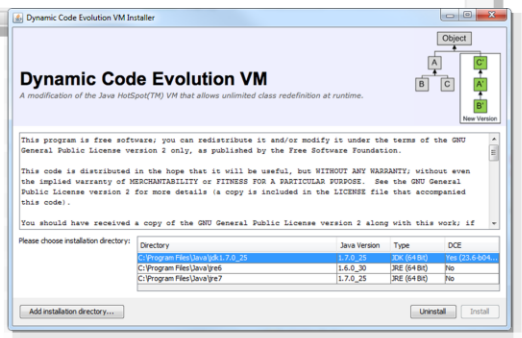



Every PCF file declares either a container widget or a location. It is not possible to create a PCF file that declares only a single isolated atomic widget.

# Internal Debug Tools for PCFs




- config.xml parameter
  - EnableInternalDebugTools
  - value="true"
- Install and enable DCEVM for JDK
  - Developer environments ONLY!

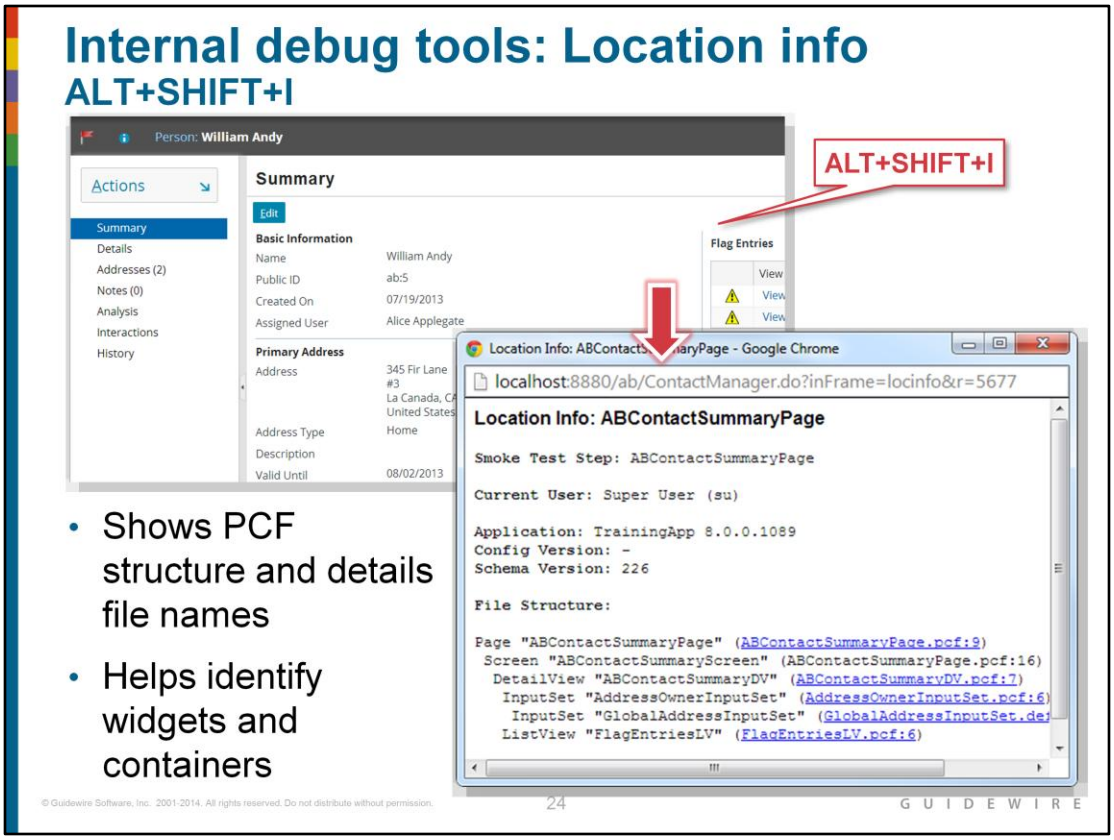






You can use the internal debug tools to identify the PCF structure, associated widgets, PCF related GOSU code, and even open the PCF in Studio. It is optional to install the DCEVM for the JDK in order to use the internal debug tools.

Guidewire recommends that you install the DCEVM in development environments for a more efficient development experience. The DCEVM allows for "hot swapping" of classes. The DCEVM should NOT be installed in production environments.

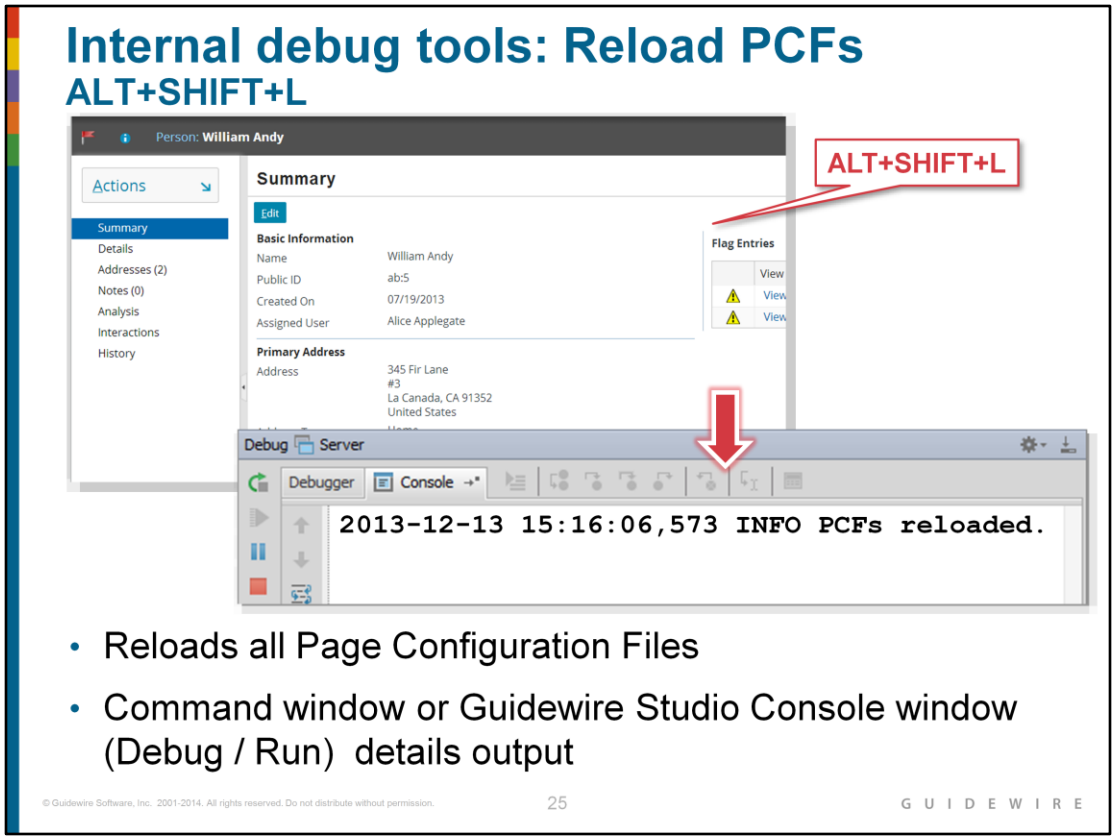



If you are running an application project with internal tools enabled, you can view the location information with ALT+SHIFT+I in a separate browser window.

The window details the file structure and details the hierarchy of the location, screen, and any child container widgets. For each location and container widget, the name of the file in which it is referenced is listed.

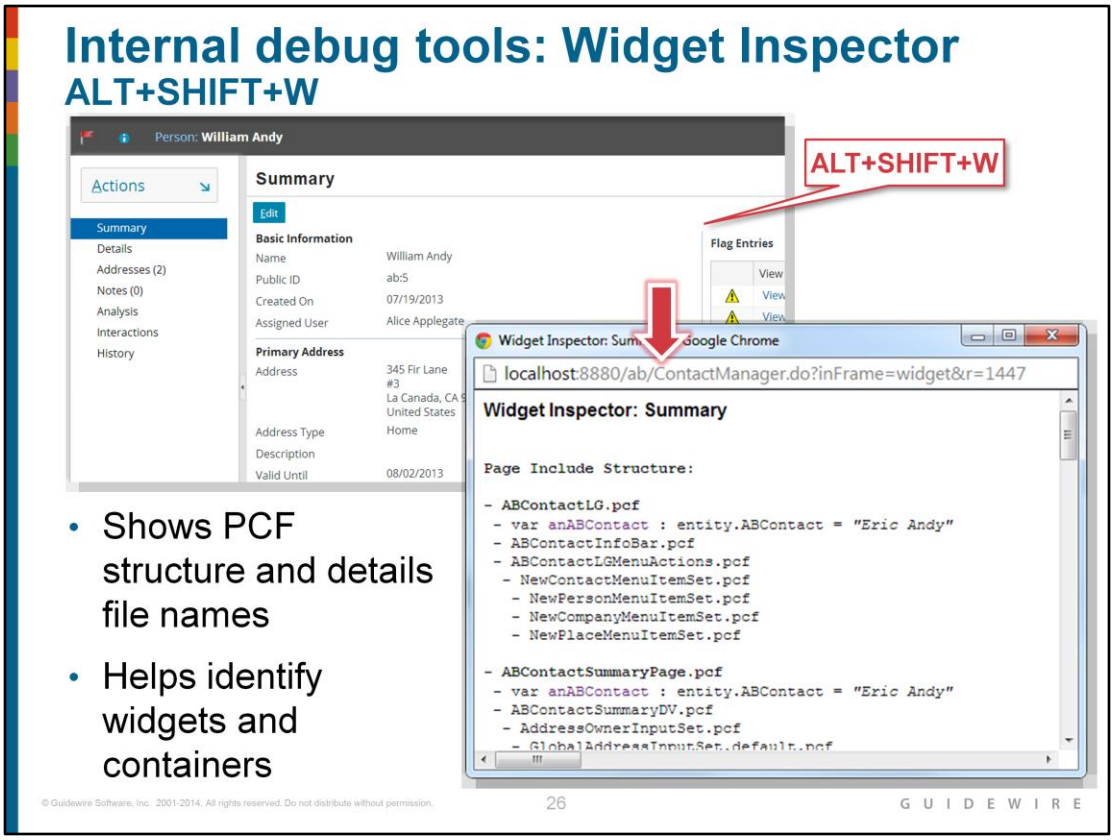
Location Info is also useful when Studio is not running or if you are viewing a modal file. Location Info identifies the mode currently displayed. Going directly to the PCF editor shows you the "first" mode until you choose another.

Location Info also shows the workspace PCF if it is defined.



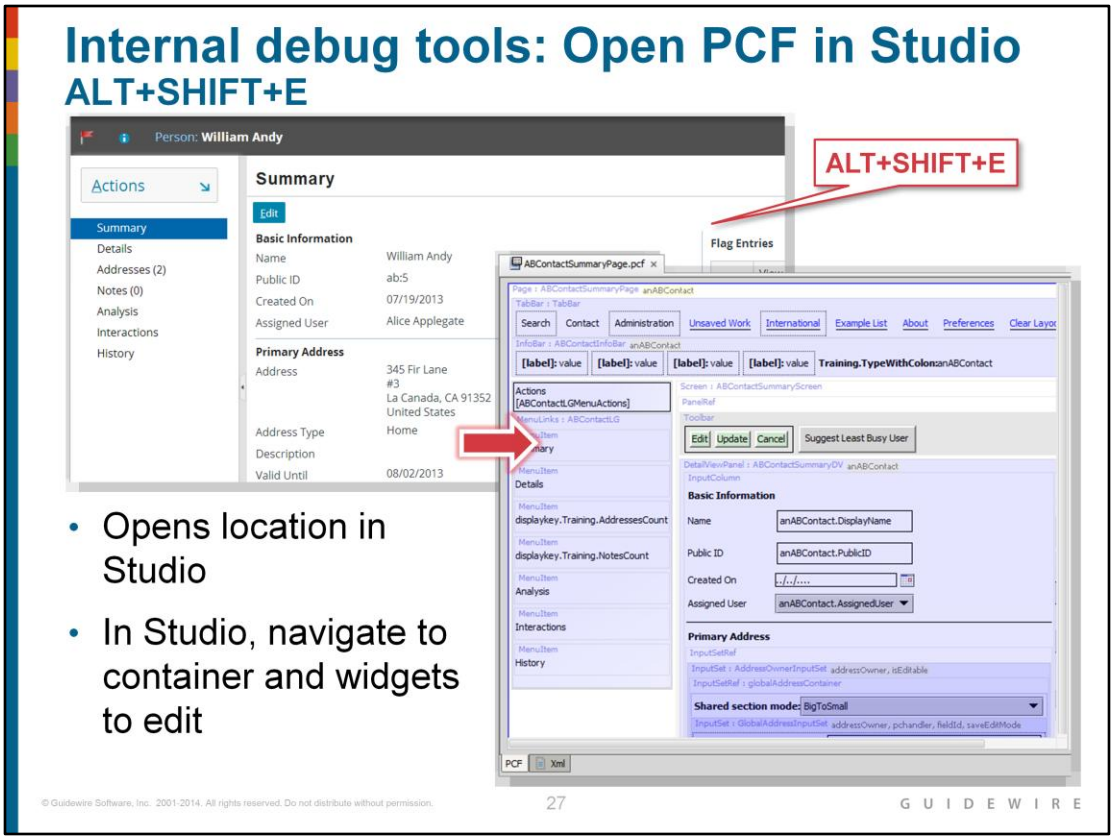

If you are running an open application project in Guidewire Studio and if internal tools are enabled, you can reload all the page configuration files and display keys for the server.

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.




If you are running an application project with internal tools enabled, you can view the widget information with ALT+SHIFT+W in a separate browser window.

The widget inspector shows all the PCF files and widgets referenced in the active application browser window except for the workspace area. To view the widget inspector, press ALT+SHIFT+W in the application.



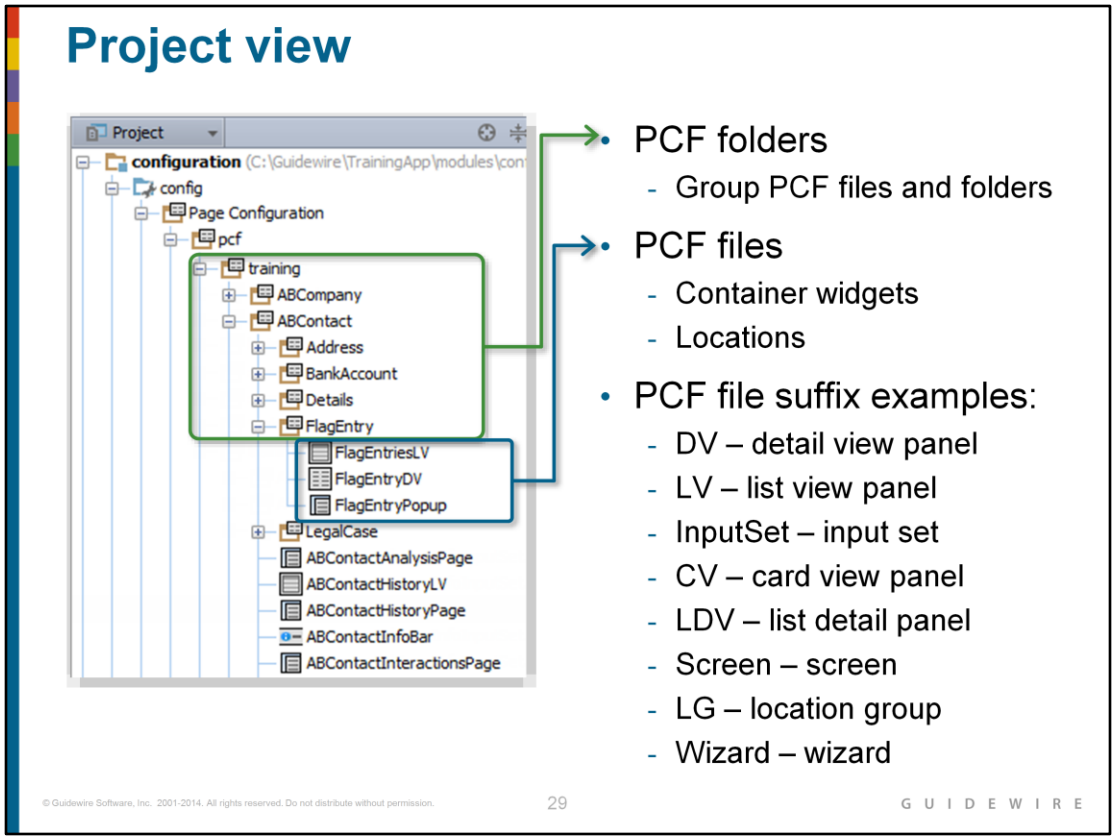

If you are running an open application project in Guidewire Studio and if internal tools are enabled, you can automatically open the location being viewed in the application. Non-wizard locations typically reference a single screen. From the screen, you can navigate in Studio to the file you wish to edit.

This technique is useful for most UI configuration in ClaimCenter and BillingCenter, where the majority of the UI configuration is done outside of wizards.

The ALT+SHIFT+E shortcut is considered experimental. While there may be circumstances where the shortcut does not function, there should never be any negative consequences in attempting to use it.

# Lesson outline

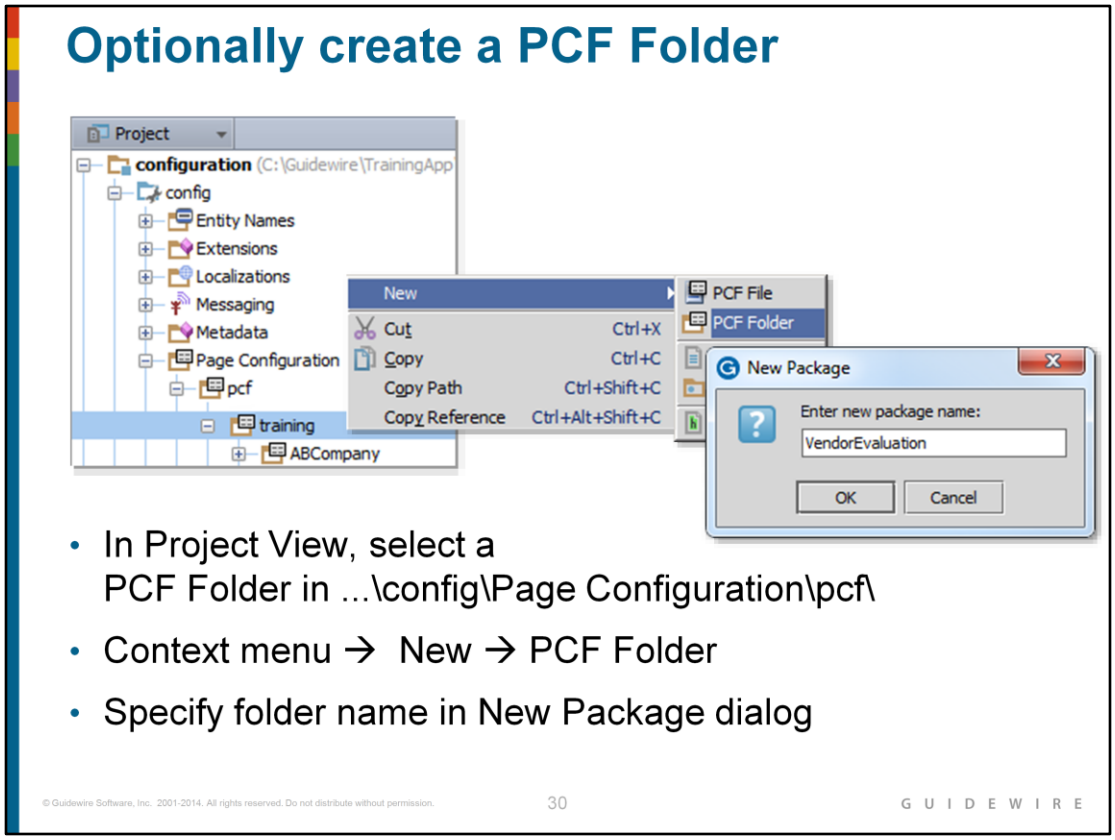
- User interface architecture
- PCF files
- Create and open PCF files
- Modify PCF files
- Deploy PCF files




Any PCF file in any folder can reference content in another PCF file in any other PCF folder. Developers often logically group related PCF files in PCF Folders.

Guidewire recommends that suffixes be used with PCF file names to indicate the contents of the file. In some cases, the suffix is required. If you create the PCF using Guidewire Studio, Studio will automatically append the file suffix for you.
Common suffixes include:

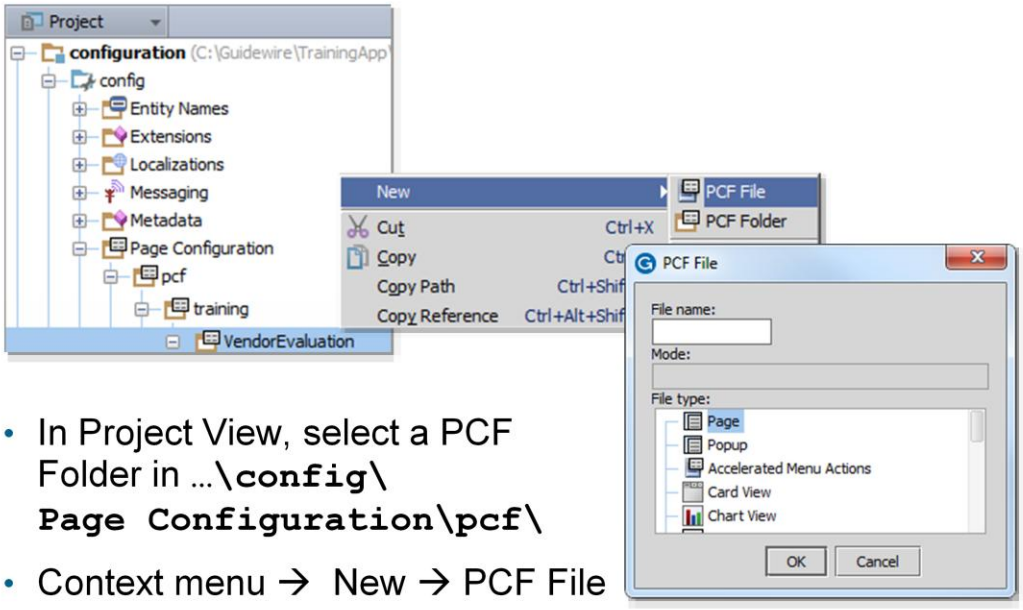
- DV – detail view panel (detail view)
- LV – list view panel (list view)
- InputSet – input set
- CV – card view panel (card view)
- LDV – list detail panel (list-detail view)
- Screen – screen
- LG – location group
- Wizard – wizard




PCF Folders organize PCF files. Consider organizing PCF Files into PCF Folders.

In the physical folders for the application, there is no Page Configuration Folder. The physical folder is "web". The Page Configuration Folder in the Project View is an alias for the physical web folder in the file system.

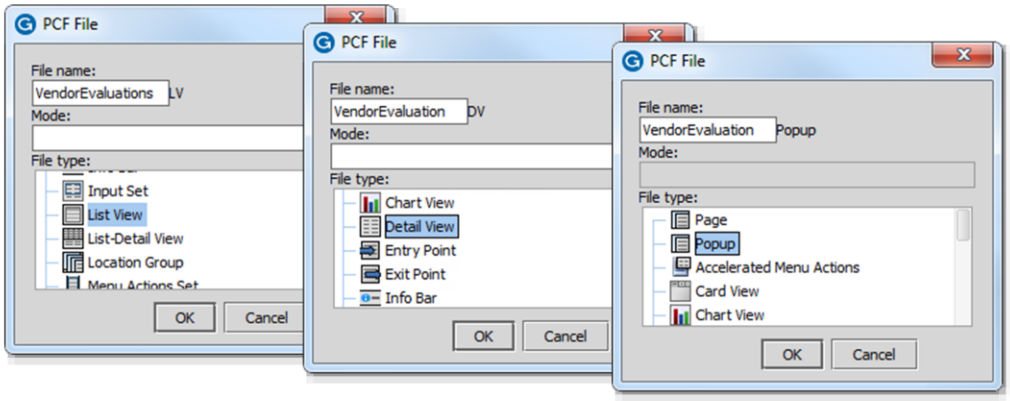
# Create a PCF File (1)




- In Project View, select a PCF Folder in …**`\config\Page Configuration\pcf\`**
- Context menu → New → PCF File
- Opens PCF File dialog

# Create a PCF File (2)

- PCF File dialog appends suffix in most cases
  - File name + File Type selection
  - Adds suffix to File Name
- Examples
  - VendorEvaluationsLV, VendorEvaluationDV, VendorEvaluationPopup



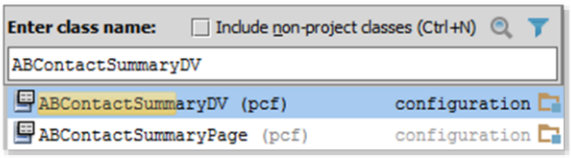




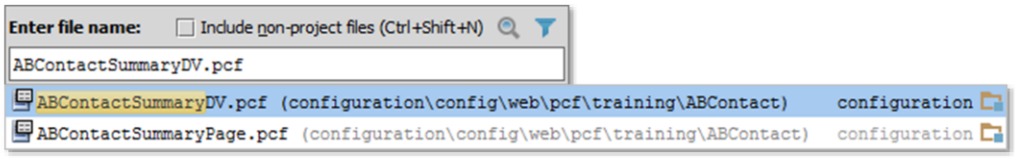

Specify the name of the file in the dialog and the file type. The dialog automatically appends the suffix to the file name when necessary.

# Open a PCF in Studio

- CTRL+N
  - Find Class…



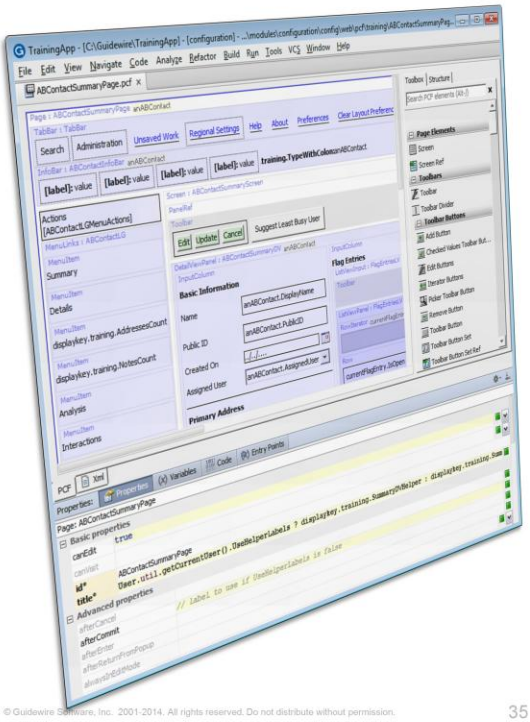

- CTRL+SHIFT+N
  - Find File…



# Lesson outline

- User interface architecture
- PCF files
- Create and open PCF files
- Modify PCF files
- Deploy PCF files

# Page Configuration (PCF) editor



- View and edit page configuration files (PCF)
  - …\config\web\pcf\
- View file as read-only XML
- Consists of
  - PCF canvas
  - Toolbox tab
  - Structure tab
  - Properties Window

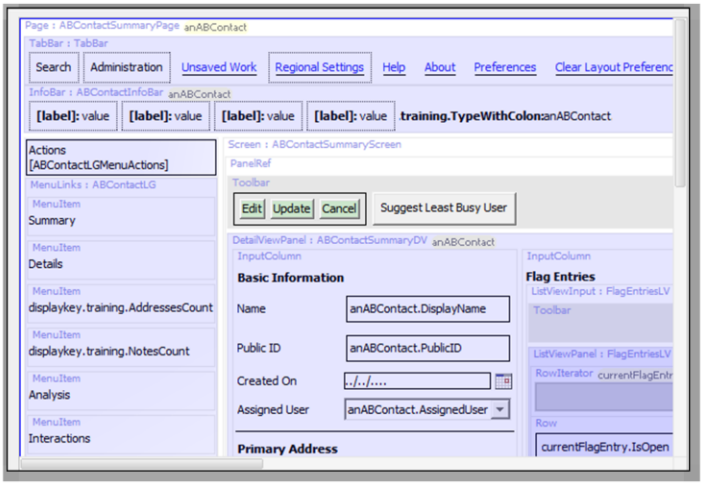




Guidewire applications use a page configuration format (PCF) files to render the application user interface. Use the Page Configuration File (PCF) editor to edit PCF files. The PCF Editor supports drag-and-drop composition of PCF pages using graphical elements, PCF page grouping views, Gosu code, display key creation, and much more.

# PCF editor: Canvas

- Graphical representation of PCF elements (widgets):
  - Content
  - Container
  - Gosu symbol
  - Conditionally visible
  - Iterator
  - RowIterator
- Colors
  - Hierarchy
  - Warnings
  - Placement




- Supports drag-and-drop from toolbox
- Context menu commands





The PCF editor contains a canvas that provides a graphic display of the contents of the file. From the canvas, you can select an item to display and edit its properties. You can also move and delete existing items. In addition, the canvas in the Studio center pane provides a drag-and-drop capability for working with the PCF elements (widgets) on the PCF page. On the canvas, Studio displays:

- Elements that represent actual screen content (inputs, and similar items) as a simplified version of how they appear within ClaimCenter.
- Elements that function primarily as containers (data view panels, for example) as light gray boxes, with a header indicating the element type and ID.
- Elements that define or expose additional Gosu symbols to their descendants as light gray boxes, with a list of symbols at the top. If you move your mouse over a symbol, Studio shows a tooltip with the name, type, and initial value of the symbol.
- If the symbol represents a Require, the tooltip indicates this as well.
- If you click a symbol name, Studio selects the containing element, and then opens the appropriate properties tab for editing whatever is providing the symbol. Finally, if necessary, Studio selects the symbol in the Properties tab.
- Elements that are conditionally visible with a dotted border.
- Elements that iterate over a set of data and produce their contents once for each element in the data by a single copy of the contents. It follows this with an ellipsis to indicate iteration.
- RowIterator widgets with inferred header and footer cells in the position in which they appear.

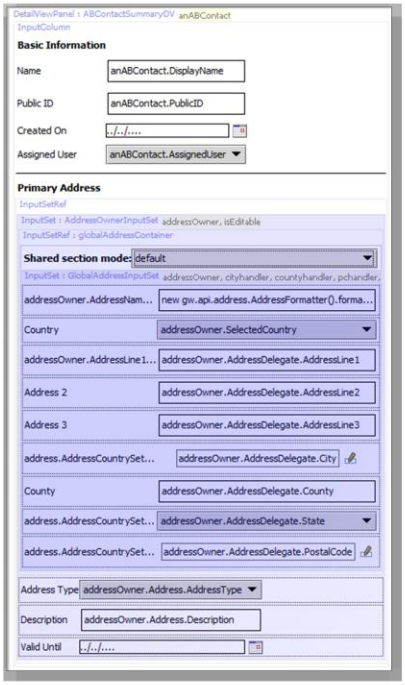
The Canvas also allows for you to view all the nested elements. Files can be nested as deep as necessary, in which case the color of the shaded area will be progressively deeper, for example:

- White - defined in this file
- Light blue -defined in other file that is referenced by this file
- Dark blue - defined in file referenced by referenced file

Double-clicking at any level will open the selected PCF directly.

# Included sections

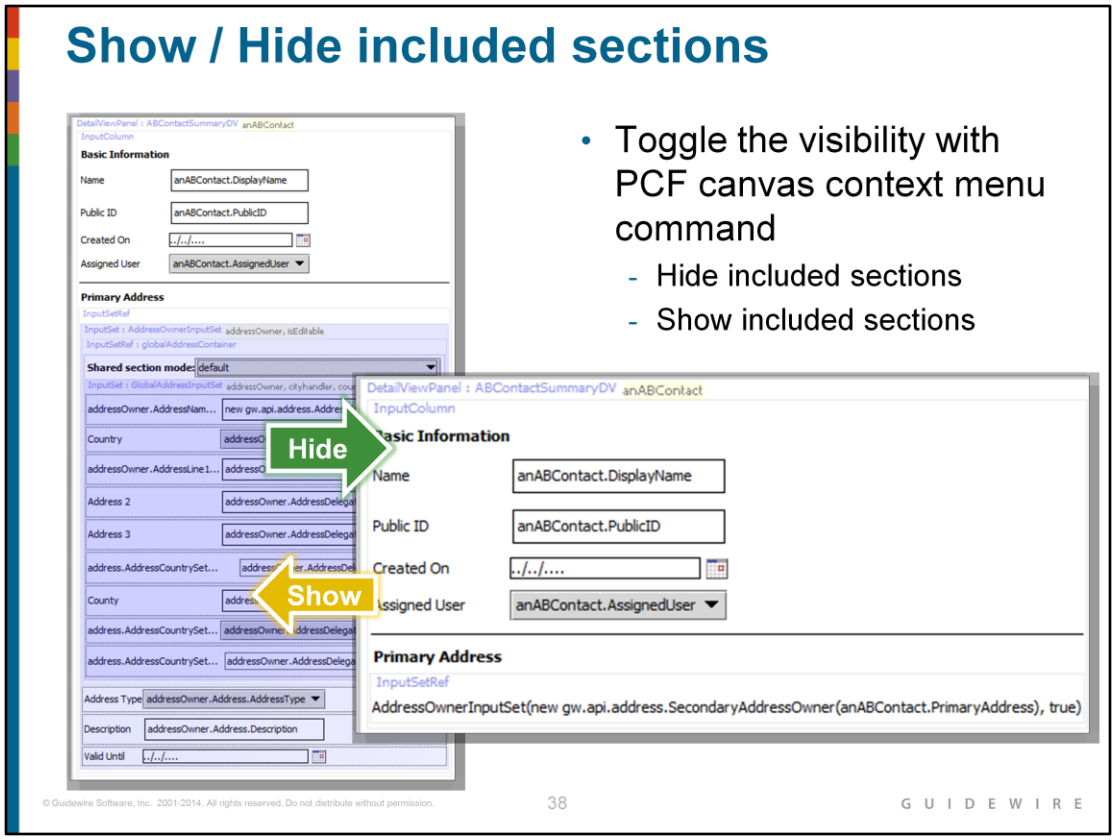
- White
  - In this file
- Light purple
  - Reference to another file
- Darker purple
  - Reference to a file that includes another reference to a file
- Color darkens with additional nesting of included sections



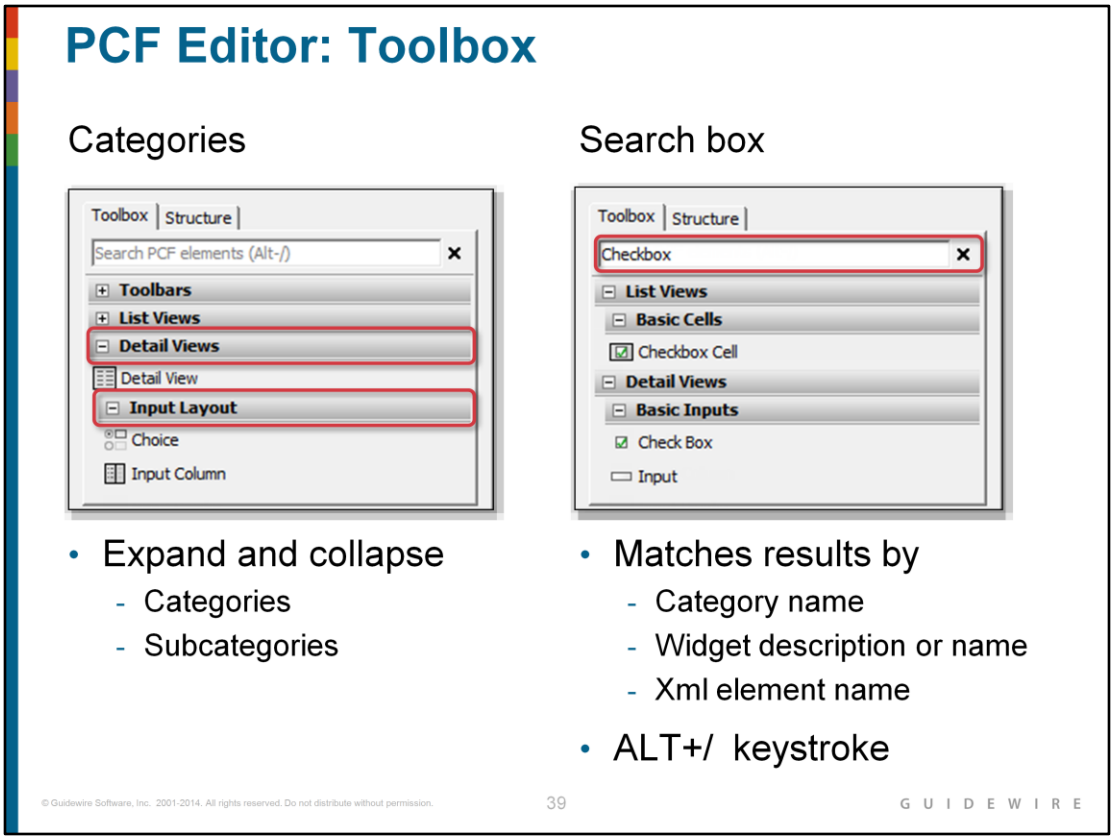





Files can be nested as deep as necessary, in which case the color of the shaded area will be progressively deeper. Double-clicking at any level will open the selected file directly. You can toggle the visibility of child files embedded in a parent PCF file. If you disable the representation of the included files, Studio displays the text of the reference expression instead.




Show/Hide included sections: Toggle the visibility of child files embedded in a parent PCF file. If you disable the representation of the included files, Studio displays the text of the reference expression instead.



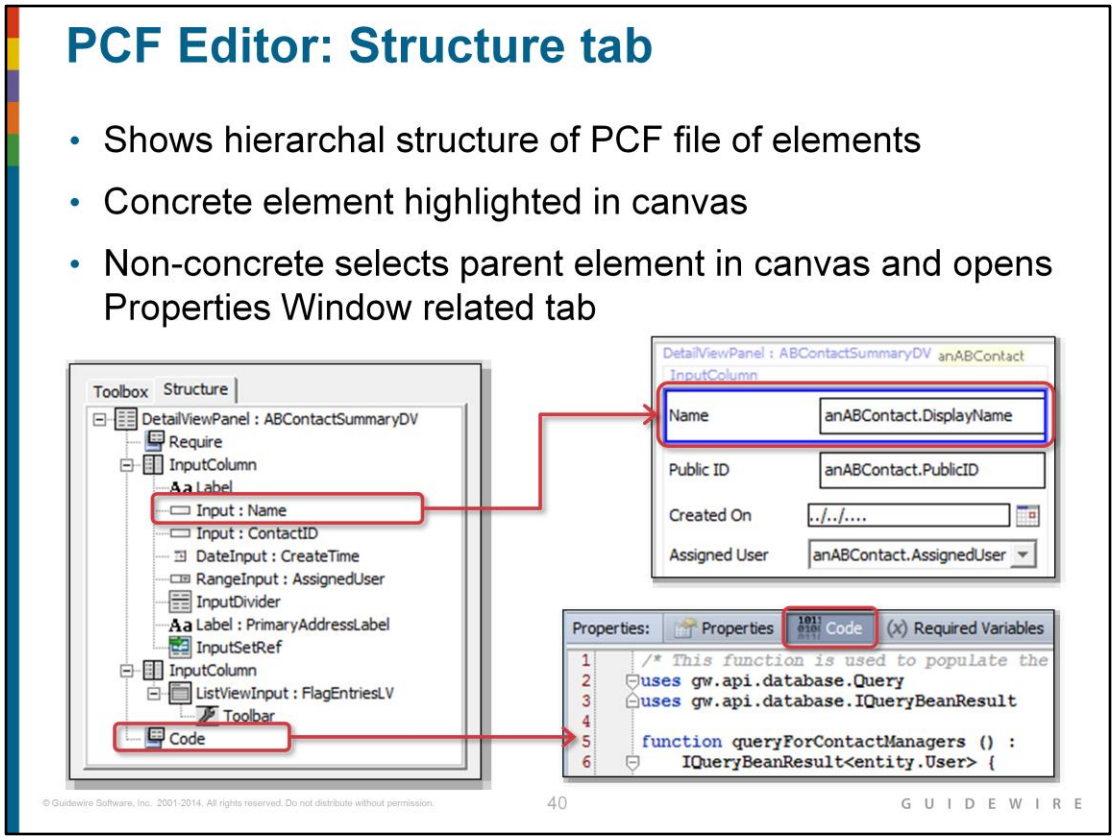

The Toolbox tab is a list of widgets. To add elements to a PCF file, you select the corresponding tool in the toolbox and drag it onto the canvas. The Toolbox tab contains a search box and a list widgets organized into specific categories and subcategories.

Clicking on a category name expands or collapses that category. Clicking on a subcategory name expands or collapses that subcategory as well. Guidewire Studio only displays widget categories containing widgets that are valid and available for use in the current PCF file. Guidewire Studio displays a description of that widget in a tooltip when you hover over the widget with your mouse.

You use the search box to filter the full set of widgets. As you type a search term into the search box, the toolbox filters the results by:

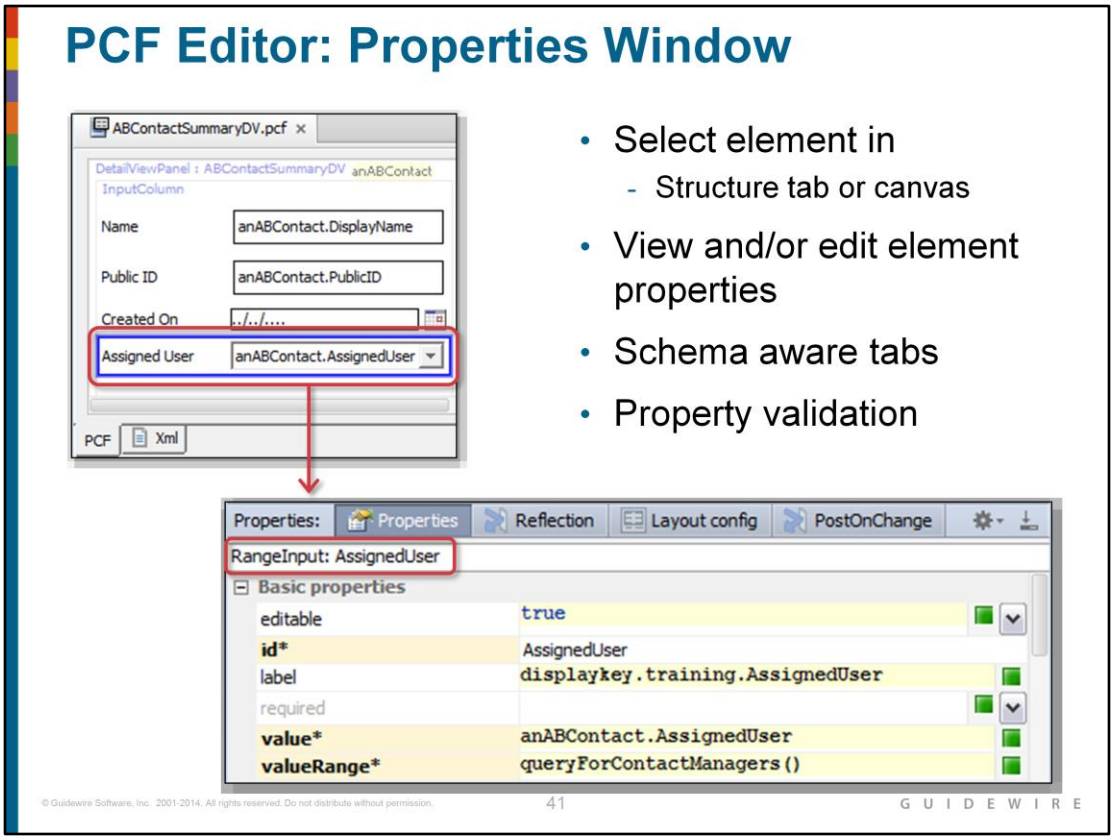
- Any widgets whose category name matches the typed text
- Any widgets whose name matches the typed text
- Any widgets whose actual name in the XML matches the typed text
- Any widgets whose description contains the typed text

Clicking the X icon by the search box clears text from the box and stops filtering the widget list. Keyboard shortcut ALT+/ gives focus to the search box.




The Structure tab shows the hierarchical structure of the PCF file as a tree. Each node in the tree represents a PCF element. Any children of the node are children of that element.
If you click an element that represents a concrete element on the canvas, Studio selects that element on the canvas.

If you click on an element that does not represent a concrete element on the canvas, then Studio first selects the containing element on the canvas. It then selects the appropriate properties tab with which to edit the clicked element. Finally, if necessary, Studio selects the clicked element in the properties tab (at the bottom of the screen).

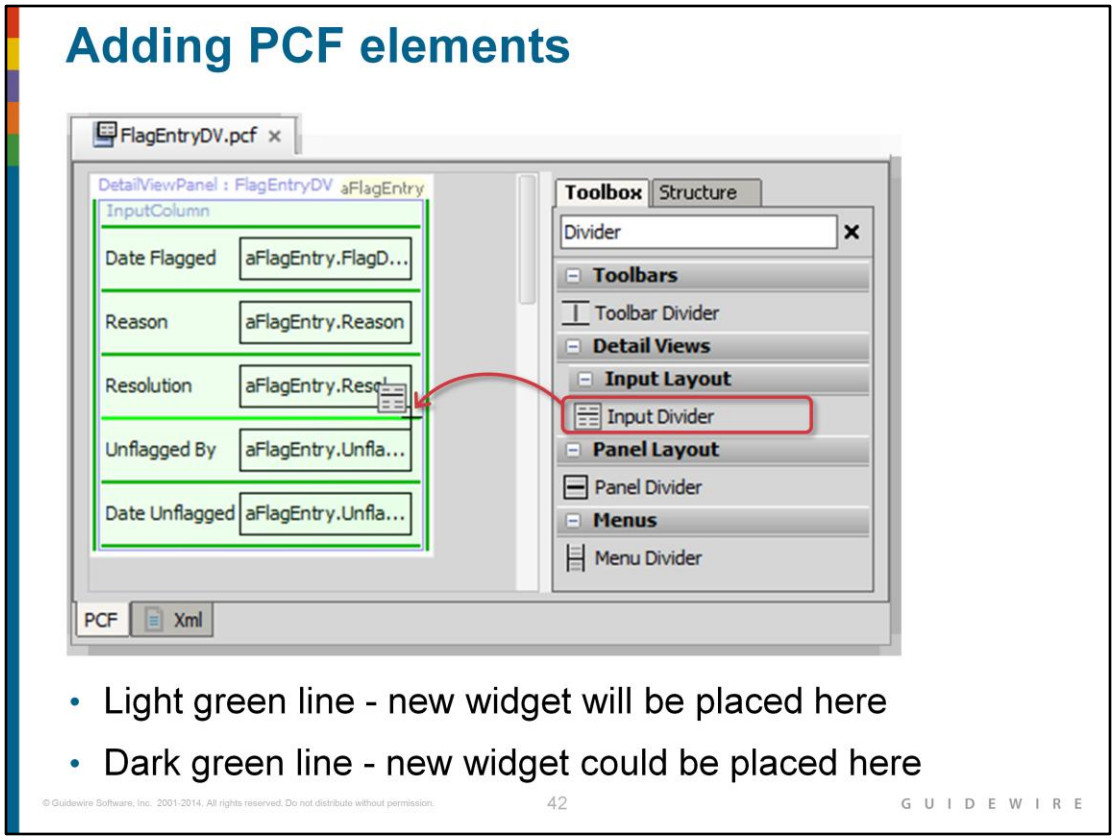



All PCF elements have definable properties in the Properties window. To view properties of a PCF file, click its title link in the upper-left corner. To view properties of any element, click that element.

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.

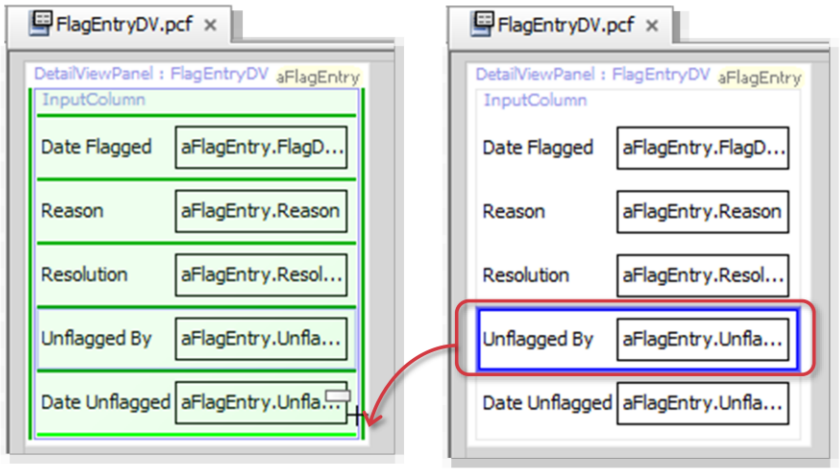
The Properties window validates each property expression and/or and value.




To add a new widget to a PCF file, select the appropriate tool from the toolbox and drag the widget onto the canvas. As soon as you drag onto the canvas, a series of green lines appear.

Dark green lines indicate places where the widget can legally be placed. A single light green line indicates the place where the widget will be placed if you release the mouse.

# Moving PCF elements




- Select element to move and drag to valid location
  - Light green line - new widget will be placed here
  - Dark green line - new widget could be placed here

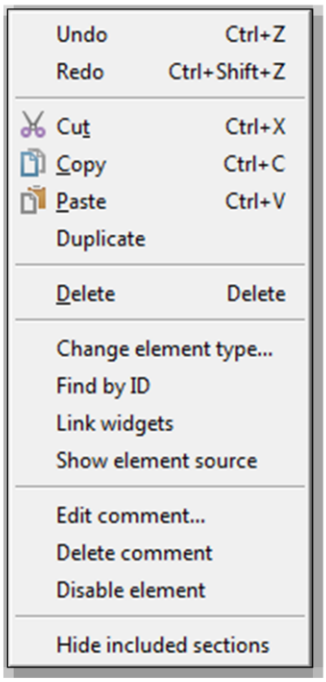




Dark green lines indicate places where the widget can legally be placed. A single light green line indicates the place where the widget will be placed if you release the mouse.

## Element context menu commands

- Standard editing commands
- Change element type…
  - Opens dialog of possible substitutes
- Find by ID
  - Opens dialog to available elements
- Link widgets
  - Create explicit iterator references
- Show element source
  - Opens window with read-only XML
- Edit comment… / Delete Comment
- Disable element
- Show/Hide included sections

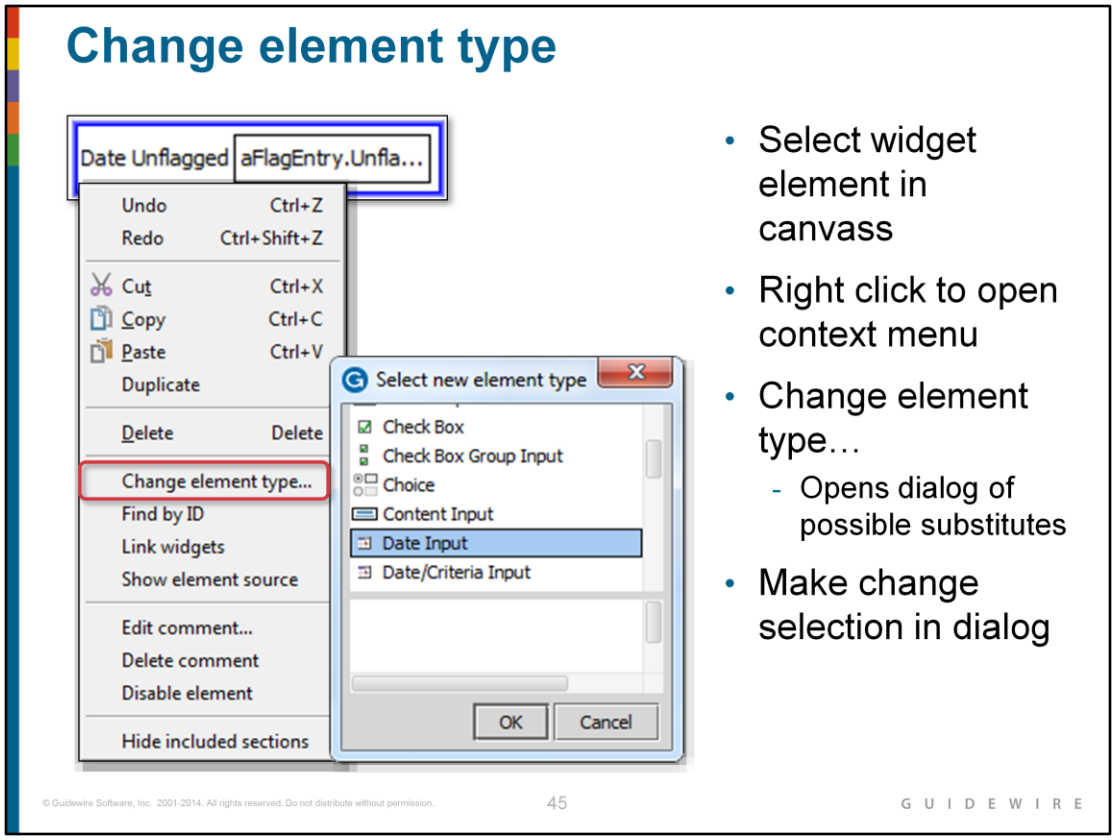






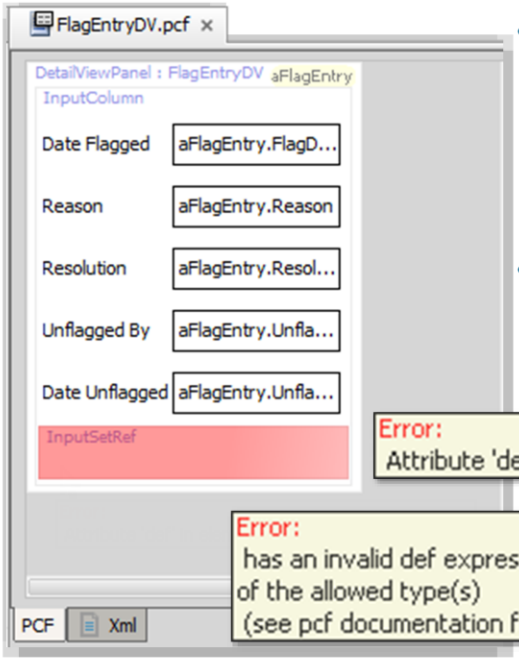

The canvas element context menu has many menu actions as explained below:

- Enable element: Enable a previously disabled element. This action removes the surrounding comment tags from the element.
- Disable element: Disable an element by commenting out the widget. This prevents ClaimCenter from rendering the widget in the interface.
- Delete comment: Remove a comment from an element.
- Edit comment...:Attach a comment to any element on the canvas
- Link widgets: Link widgets on a parent page that spans multiple child PCF files. You use this particularly for explicit iterator references.
- Show/Hide included sections: Toggle the visibility of child files embedded in a parent PCF file. If you disable the representation of the included files, Studio displays the text of the reference expression instead.
- Show element source: View the XML code for an element. Studio displays the XML code in a pop-up window.
- Find by ID: Find an element by its ID. The dialog contains a filter text field.  The dialog lists all elements on the canvas that have their id attribute set.




"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.

# Error messaging




- Editor red highlights the widget element with an error
  - Misconfiguration
  - Missing configuration
- Mouse over element to see Error Message



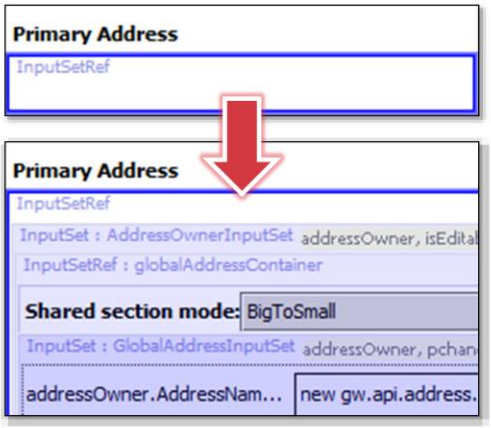


The PCF editor canvas highlights elements with errors.

# Refresh PCF


- Refresh the display of the current PCF file
  - **Main toolbar → Refresh PCF**

- Helps with widgets that include another widget by reference
- Examples
  - InputSetRef
  - PanelRef
  - ScreenRef



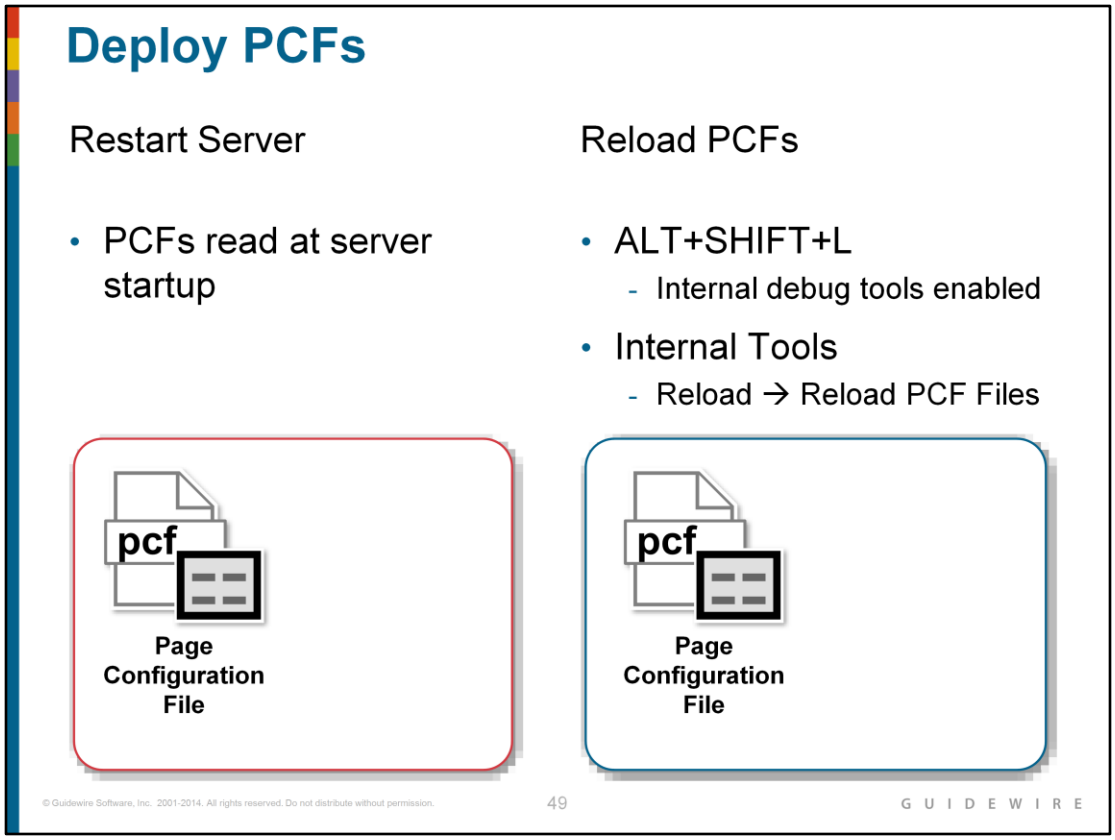




When you add a PCF widget or edit a PCF widget that includes by reference another widget, you can Refresh the PCF to display the referenced widget.

# Lesson outline

- User interface architecture
- PCF files
- Create and open PCF files
- Modify PCF files
- Deploy PCFs




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:
  - Describe the user interface architecture for Guidewire applications
  - Describe the kinds of Page Configuration Files (PCF)
  - Create new PCF folders and files
  - Open and edit PCF files
  - Deploy PCF files

# Review questions

1. Define the broad purpose for each element category:
   a) Atomic widgets
   b) Container widgets
   c) Locations
2. What does a PCF file define?
3. What does each PCF editor convention mean?
   a) A container widget shaded light blue
   b) A container widget shaded dark blue
   c) A light green line
   d) A dark green line
   e) A red background
4. How do you deploy PCF changes?




Answers
1a) Atomic widgets display individual elements of data and/or functionality.
1b) Container widgets group atomic widgets into logical groups.
1c) Locations define how users move from one place in the application to the next.
2) A PCF file defines a container widget or location and its contents.
3a) The container widget is referenced by the current PCF file and Its contents are contained in another file
3b) The container widget is referenced by some container widget referenced by the current PCF file. It is in a "grandchild" file.
3c) For the currently selected widget tool, a new widget of that type will be placed in the current location.
3d) For the currently selected widget tool, a new widget of that type could be placed in the current location.
3e) The PCF file has an error in it.
4) You can Restart the server. Restarting the server reloads PCF files at startup. A more efficient approach for development is to Reload the PCF files. With internal tools enabled, you can use the ALT+SHIFT+L keystroke to reload the PCF files. You can also log in with an administrator user account and reload PCF files from the Internal Tools Reload page.

# Notices