



# **InsuranceSuite Fundamentals Essentials**

## **Student Workbook**

# Table of Contents

|  |           |
|--|-----------|
| <b>Introduction .....</b>  | <b>5</b>  |
| <b>Lesson 1 Creating New Entities.....</b>                               | <b>6</b>  |
| 1.1 Prerequisites.....   | 6         |
| 1.2 Lab: Create a new entity.....  | 6         |
| 1.2.1 Verification .....   | 7         |
| 1.3 Lab: Define an array relationship between two entities.....          | 8         |
| 1.3.1 Verification .....   | 9         |
| 1.4 Lab Solution: Create a new entity.....                               | 9         |
| 1.5 Lab Solution: Define an array relationship between two entities..... | 13        |
| <b>Lesson 2 List Views .....</b>   | <b>16</b> |
| 2.1 Prerequisites.....   | 16        |
| 2.2 Lab: Create a List View .....  | 17        |
| 2.3 Lab: Reference the List View.....                                    | 18        |
| 2.3.1 Verification .....   | 18        |
| 2.4 Bonus lab: List Views and navigation.....                            | 20        |
| 2.4.1 Verification .....   | 20        |
| 2.5 Lab Solution: Create a List View .....                               | 22        |
| 2.6 Lab Solution: Reference the List View .....                          | 28        |
| 2.7 Bonus Lab Solution: List Views and navigation.....                   | 30        |
| <b>Lesson 3 Editable List Views .....</b>                                | <b>31</b> |
| 3.1 Prerequisites.....   | 31        |
| 3.2 Lab: Make the Interactions List View editable .....                  | 32        |
| 3.2.1 Verification .....   | 32        |
| 3.3 Lab Solution: Make the Interactions List View editable .....         | 34        |
| <b>Lesson 4 Typelists .....</b>  | <b>39</b> |
| 4.1 Prerequisites.....   | 40        |
| 4.2 Configuration .....  | 40        |
| 4.3 Lab: Create new typelists .....                                      | 40        |
| 4.4 Lab: Add typekey fields to the entity.....                           | 41        |
| 4.4.1 Verification .....   | 42        |
| 4.5 Lab Solution: Create new typelists .....                             | 43        |
| 4.6 Lab Solution: Add typekey fields to an entity.....                   | 46        |
| <b>Lesson 5 Popups: View and Edit.....</b>                               | <b>48</b> |
| 5.1 Prerequisites.....   | 48        |

|   |           |
|---|-----------|
| 5.2 Lab: Popup locations .....  | 49        |
| 5.2.1 Verification .....  | 51        |
| 5.3 Lab Solution: Popup locations .....                                     | 53        |
| <b>Lesson 6 Validation.....</b>   | <b>59</b> |
| 6.1 Prerequisites.....  | 59        |
| 6.2 Lab: Field-level validation in the UI.....                              | 59        |
| 6.2.1 Investigation.....  | 59        |
| 6.2.2 Configuration .....   | 60        |
| 6.2.3 Verification .....  | 61        |
| 6.3 Lab: Field-level validation in the Data Model.....                      | 63        |
| 6.3.1 Configuration .....   | 63        |
| 6.3.2 Verification .....  | 64        |
| 6.4 Lab: Validation rules .....   | 65        |
| 6.4.1 Configuration .....   | 65        |
| 6.4.2 Verification .....  | 66        |
| 6.5 Lab Solution: Field-level validation in the UI .....                    | 67        |
| 6.6 Lab Solution: Field-level validation in the Data Model.....             | 69        |
| 6.7 Lab Solution: Validation rules .....                                    | 70        |
| <b>Lesson 7 Input Sets .....</b>  | <b>73</b> |
| 7.1 Prerequisites.....  | 74        |
| 7.2 Lab: Create an Input Set .....  | 74        |
| 7.3 Lab: Reference the Input Set.....                                       | 75        |
| 7.3.1 Verification .....  | 76        |
| 7.4 Lab Solution: Create an Input Set .....                                 | 79        |
| 7.5 Lab Solution: Reference the Input Set .....                             | 82        |
| <b>Lesson 8 Partial Page Update.....</b>                                    | <b>85</b> |
| 8.1 Prerequisites.....  | 85        |
| 8.2 Lab: Targeted Post On Change – DATA_ONLY .....                          | 86        |
| 8.2.1 Configuration .....   | 86        |
| 8.2.2 Verification .....  | 87        |
| 8.3 Lab: Advanced targeted Post On Change – layout re-render .....          | 88        |
| 8.3.1 Investigation.....  | 88        |
| 8.3.2 Configuration .....   | 89        |
| 8.3.3 Verification .....  | 89        |
| 8.4 Lab Solution: Configure targeted Post On Change – DATA_ONLY .....       | 91        |
| 8.5 Lab Solution: Advanced targeted Post On Change – layout re-render ..... | 92        |
| <b>Lesson 9 Subtypes.....</b>   | <b>96</b> |
| 9.1 Prerequisites.....  | 96        |

|   |            |
|---|------------|
| 9.2 Lab: Create a new custom subtype entity .....                         | 96         |
| 9.3 Lab: Extend an existing base application subtype entity.....          | 98         |
| 9.3.1 Verification .....  | 99         |
| 9.4 Lab Solution: Create a new custom subtype entity .....                | 100        |
| 9.5 Lab Solution: Extend an existing base application subtype entity..... | 102        |
| <b>Lesson 10 Modes .....</b>  | <b>104</b> |
| 10.1 Prerequisites.....   | 104        |
| 10.2 Lab: Create a set of modal Input Sets.....                           | 104        |
| 10.3 Lab: Reference the set of modal Input Sets .....                     | 106        |
| 10.3.1 Verification .....   | 107        |
| 10.4 Lab Solution: Create a set of modal Input Sets.....                  | 109        |
| 10.5 Lab Solution: Reference the set of modal Input Sets.....             | 115        |
| <b>Lesson 11 Entity Names.....</b>  | <b>117</b> |
| 11.1 Prerequisites.....   | 117        |
| 11.2 Lab: Create and reference an entity name.....                        | 118        |
| 11.2.1 Verification .....   | 119        |
| 11.3 Lab Solution: Create and reference an Entity Name.....               | 121        |

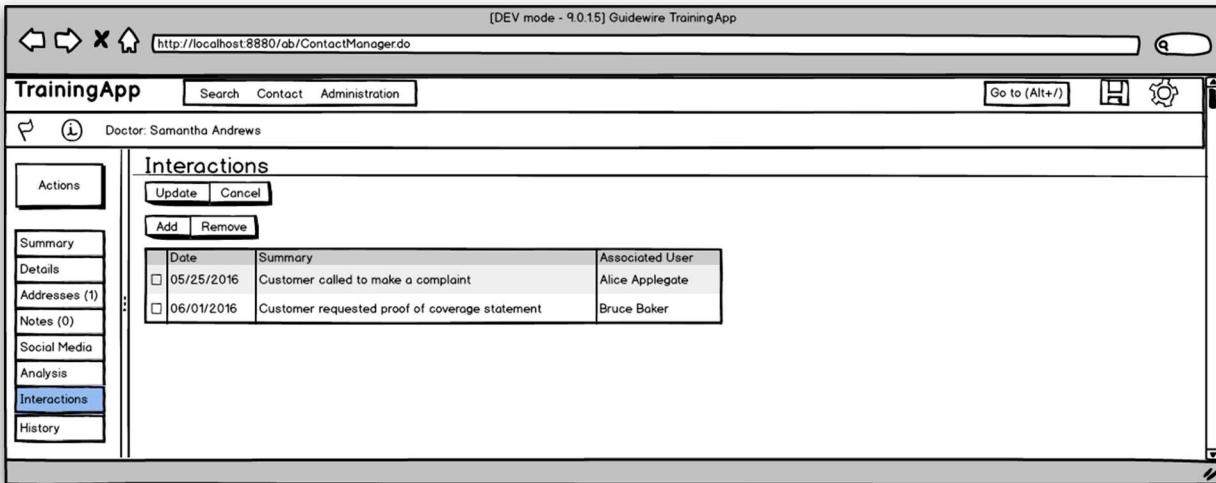
# Introduction

Welcome to the Guidewire InsuranceSuite Fundamentals Essentials course.

The Student Workbook will lead you through the course labs. The lesson numbers correspond to the lesson numbers in your training. Complete the assigned labs to the best of your ability.

# Lesson 1 Creating New Entities

The business analysts want to capture specific information about each contact interaction such as a telephone call, email message, postal mail letter, or in-person office visit. Eventually, they want the information displayed in an Interactions section of TrainingApp.



The screenshot shows a web browser window for 'Guidewire TrainingApp' in DEV mode. The URL is http://localhost:8880/ab/ContactManager.do. The main content area displays the 'Interactions' section for a contact named 'Doctor: Samantha Andrews'. On the left, there's a sidebar with links: Actions, Summary, Details, Addresses (1), Notes (0), Social Media, Analysis, Interactions (selected), and History. The 'Interactions' section has buttons for Update, Cancel, Add, and Remove. A table lists two interactions:

| Date       | Summary  | Associated User |
|------------|--|-----------------|
| 05/25/2016 | Customer called to make a complaint            | Alice Applegate |
| 06/01/2016 | Customer requested proof of coverage statement | Bruce Baker     |

*"We want a CSR (Customer Service Representative) such as Alice Applegate to be able to create and edit details about contact interactions."* – Insurance company business analysts

In this lab, your job is to make the necessary **data model changes** to meet customer requirements. As a configuration developer, you will use the Entity Editor in Guidewire Studio to modify the TrainingApp data model. You will implement the **user interface changes** in a later lesson.

## 1.1 Prerequisites

## 1.2 Lab: Create a new entity

As a configuration developer, you want to be able to create new entities. In this lab, you will use the Entity Editor to create an entity to capture contact interaction details.

1. Create a new entity named `Interaction_Ext` and add the following fields:

| Field Name         | Datatype                        | Null ok? |
|--------------------|---------------------------------|----------|
| InteractionDate    | date and time                   | true     |
| InitiatedByContact | A boolean value                 | true     |
| Summary            | A string of up to 60 characters | true     |
| AssociatedUser     | A foreign key to User           | true     |

### 1.2.1 Verification

2. Use Studio's code generation feature to generate the Java class from the entity and verify the generated Java class
  - a) If possible, use incremental code generation
  - b) Verify that there were no errors during code generation
  - c) Open the generated Java class



#### Important!

Read carefully.

3. For each new entity element, remember to set the `nullok` attribute to true if not specifically defined to be false.
4. When required, add an element description and specify column parameters



## Best Practices

Use \_Ext in the entity name

Notice that the name of the entity has an \_Ext suffix. For new entities, Guidewire recommends that the name of the entity should end with \_Ext (or start with Ext\_). This is to prevent potential conflicts during the upgrade to the next version of the Guidewire application.

## 1.3 Lab: Define an array relationship between two entities

As a configuration developer, you want to be able to define different types of relationships between entities. In this lab, you will define an array relationship between ABContact and Interaction\_Ext:

"An ABContact entity can have zero or many Interactions associated with it." – Insurance company business analysts

### 1. Add an array key field to ABContact

- d) Extend the ABContact entity to define a new array. The array key field should point to Interaction\_Ext. The array name should be descriptive and should follow the recommended naming conventions.

### 2. Use Studio's code generation feature to generate the Java class from the entity and verify the generated Java class

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Open the generated Java classes and verify that you can see the new fields
- d) Remember, for every data model array the framework automatically generates `addTo<ArrayName>(element)` and `removeFrom<ArrayName>(element)` functions. Verify that you can find those methods in the generated Java class.



## Best Practices

Use \_Ext in entity field name when extending base application entities

For fields that are added to a base application entity, Guidewire recommends that the field name should

end with \_Ext (or start with Ext\_). This is to prevent potential conflicts during the upgrade to the next version of the Guidewire application.

### 1.3.1 Verification



#### Activity

Verify the work you have done

As a configuration developer, you want to be able to properly deploy new and changed data model resources.

**1. Restart the server to deploy the changes**

- a) During server restart Studio first runs the code generators, then compiles the project and finally deploys the resources

**2. Regenerate the Data Dictionary**

- a) Follow the steps to regenerate the Data Dictionary

**3. Open the Data Dictionary**

- a) In Windows Explorer, navigate to the data dictionary.
- b) Open the data dictionary using your preferred browser.

**4. View the Interaction\_Ext entity**

- a) Verify each new field and associated datatype.

**5. View the ABContact entity**

- a) Verify you can see the new array



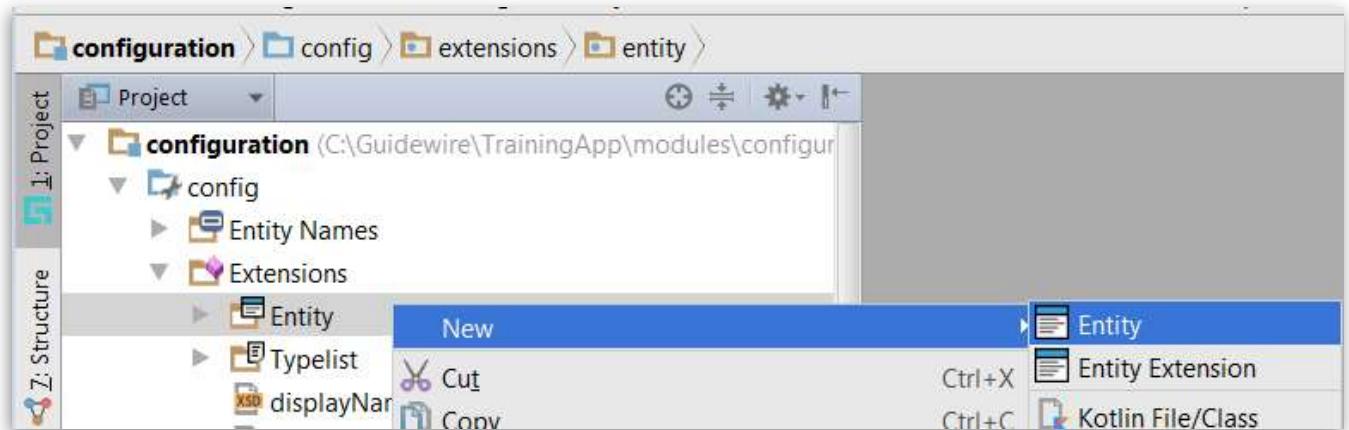
### 1.4 Lab Solution: Create a new entity

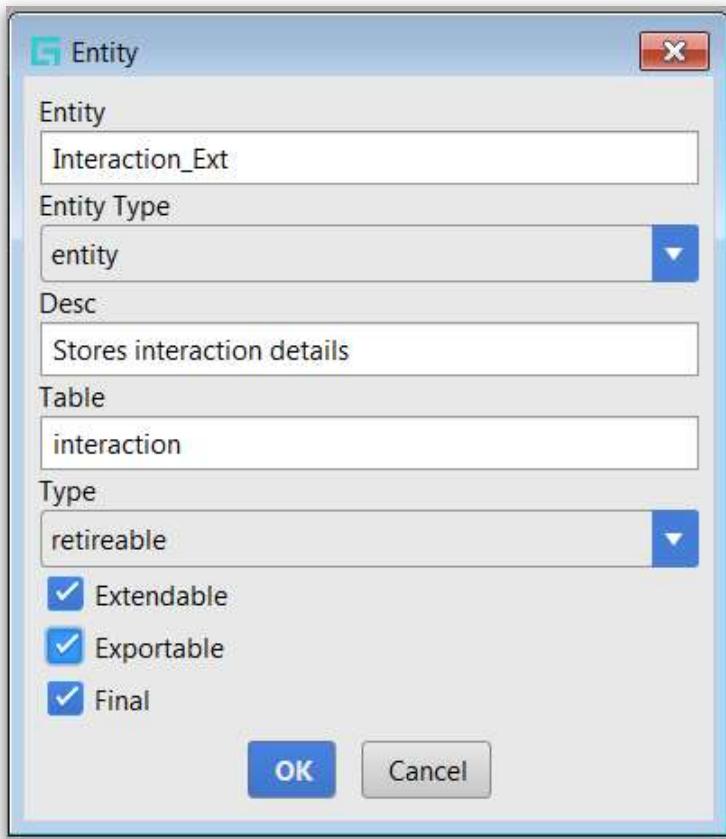


## Solution

Exact details on how to complete the lab.

### 1. Create the Interaction\_Ext entity





## 2. Add new elements to Interaction\_Ext.eti

- Add the InteractionDate field

The screenshot shows the 'Interaction\_Ext.eti' element editor. The top bar includes a 'column' button, a search icon, a minus sign, a 'Show All' dropdown, and other toolbar icons. The main area displays a table with two rows. The first row has columns 'Element', 'Primary Value', and 'Secondary Value'. The second row has columns 'Element', 'Name', and an empty column. The first row's 'Element' is 'entity', 'Primary Value' is 'Interaction\_Ext', and 'Secondary Value' is 'Stores interactio...'. The second row's 'Element' is 'column', 'Name' is 'InteractionDate', and the empty column has 'datetime' in its 'Primary Value' cell. The right side of the table shows the values for 'name' (InteractionDate), 'type' (datetime), and 'nullok' (true).

| Element | Primary Value   | Secondary Value      | Name   |                 |
|---------|-----------------|----------------------|--------|-----------------|
| entity  | Interaction_Ext | Stores interactio... | name   | InteractionDate |
| column  | InteractionDate | datetime             | type   | datetime        |
|         |                 |                      | nullok | true            |

b) Add the InitiatedByContact field

| Interaction_Ext.eti |                 |                      |                           |
|---------------------|-----------------|----------------------|---------------------------|
| Element             | Primary Value   | Secondary Value      | Name                      |
| entity              | Interaction_Ext | Stores interactio... | <b>name</b>               |
|                     | column          | InteractionDate      | bit                       |
|                     | column          | InitiatedByContact   | <b>InitiatedByContact</b> |
|                     |                 |                      | <b>type</b>               |
|                     |                 |                      | bit                       |
|                     |                 |                      | <b>nullok</b>             |
|                     |                 |                      | true                      |

c) Add the Summary field and its columnParam

| Interaction_Ext.eti |                 |                      |                             |
|---------------------|-----------------|----------------------|-----------------------------|
| Element             | Primary Value   | Secondary Value      | Name                        |
| entity              | Interaction_Ext | Stores interactio... | <b>name</b>                 |
|                     | column          | InteractionDate      | varchar                     |
|                     | column          | InitiatedByContact   | true                        |
|                     | column          | Summary              | <b>Summary</b>              |
|                     | columnParam     | varchar              | datetime                    |
|                     |                 |                      | <b>desc</b>                 |
|                     |                 | 60                   | allowInitialValueForUpgrade |
|                     |                 |                      | false                       |

d) Add the AssociatedUser field

| Interaction_Ext.eti |                 |                      |                   |
|---------------------|-----------------|----------------------|-------------------|
| Element             | Primary Value   | Secondary Value      | Name              |
| entity              | Interaction_Ext | Stores interactio... | <b>name</b>       |
|                     | column          | InteractionDate      | User              |
|                     | column          | InitiatedByContact   | true              |
|                     | column          | Summary              | AssociatedUser    |
|                     | foreignkey      | AssociatedUser       | AssociatedUserID  |
|                     |                 |                      | <b>fkentity</b>   |
|                     |                 |                      | <b>columnName</b> |
|                     |                 |                      | desc              |

3. Entities support incremental code generation. You can click on the validate icon or use the CTRL + S keystroke to kick off incremental code generation. You can open the generated Java class once the code generation is completed.

**Note:** The code generation was successful if the Codegen tool window has nothing to show.

Screenshot of some of the Java code generated by the Codegen tool.



```

configuration > generated > entity > Interaction_Ext
Interaction_Ext.java x Interaction_Ext.eti x

6  /*
7   * @ajax.annotation.Generated(value = "com.guidewire.pl.metadata.codegen.Codegen", comments = "Interaction_Ext.eti;Interaction_Ext")
8   * /deprecation, unchecked/
9   * @gw.internal.gosu.parser.ExtendedType
10  * @gw.lang.SimplePropertyProcessing
11  * @gw.entity.EntityName(value = "Interaction_Ext")
12  */
13  public class Interaction_Ext extends com.guidewire.pl.persistence.code.BeanBase implements entity.Retireable {
14      public static final gw.pl.persistence.type.EntityTypeReference<entity.Interaction_Ext> TYPE = new com.guidewire.commons.
15      public static final gw.pl.persistence.type.EntityPropertyInfoReference<gw.entity.ILink PropertyInfo> ABCONTACT_PROP = new
16      public static final gw.pl.persistence.type.EntityPropertyInfoReference<gw.entity.ILink PropertyInfo> ASSOCIATEDUSER_PROP
17
18

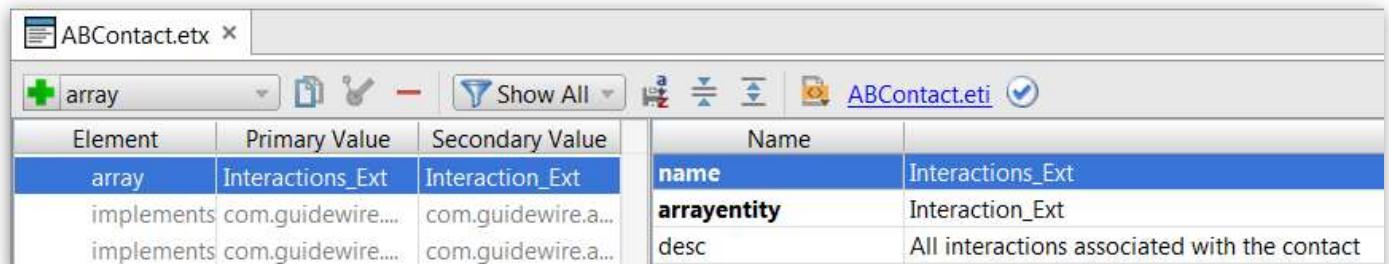
```

## 1.5 Lab Solution: Define an array relationship between two entities



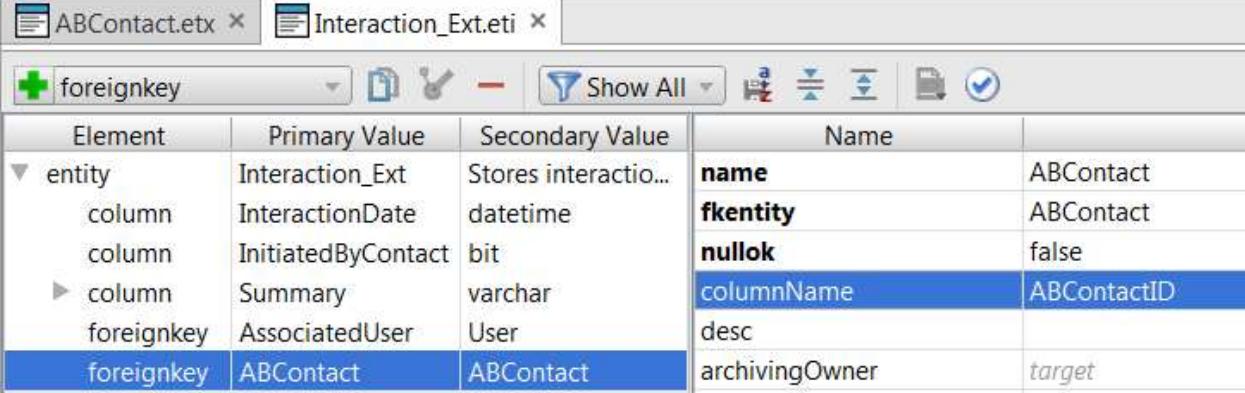
Exact details on how to complete the lab.

1. Add the array key to the ABContact.etx



| Element    | Primary Value     | Secondary Value    | Name        |  |
|------------|-------------------|--------------------|-------------|--|
| array      | Interactions_Ext  | Interaction_Ext    | name        | Interactions_Ext                             |
| implements | com.guidewire.... | com.guidewire.a... | arrayentity | Interaction_Ext                              |
| implements | com.guidewire.... | com.guidewire.a... | desc        | All interactions associated with the contact |

2. You must also add a foreign key to Interaction\_Ext.eti since every array requires a reverse foreign key on the other side of the relationship.

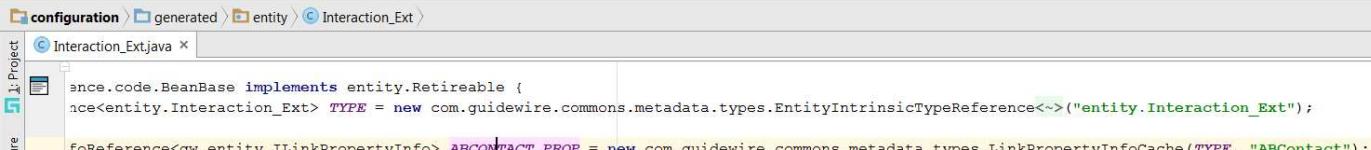


The screenshot shows the ABContact.etx configuration interface. At the top, there are tabs for 'ABContact.etx' and 'Interaction\_Ext.eti'. Below the tabs is a toolbar with icons for creating, deleting, and validating. A search bar labeled 'Show All' is followed by several filter and sort icons. The main area is a table titled 'foreignkey' with columns: Element, Primary Value, Secondary Value, Name, and Description.

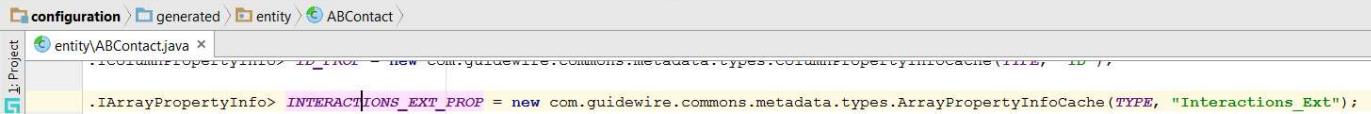
| Element    | Primary Value      | Secondary Value      | Name              |               |
|------------|--------------------|----------------------|-------------------|---------------|
| entity     | Interaction_Ext    | Stores interactio... | <b>name</b>       | ABContact     |
| column     | InteractionDate    | datetime             | <b>fkentity</b>   | ABContact     |
| column     | InitiatedByContact | bit                  | <b>nullok</b>     | false         |
| ▶ column   | Summary            | varchar              | <b>columnName</b> | ABContactID   |
| foreignkey | AssociatedUser     | User                 | desc              |               |
| foreignkey | ABContact          | ABContact            | archivingOwner    | <i>target</i> |

- Entities support incremental code generation. You can click on the validate icon or use the CTRL + S keystroke to kick off incremental code generation. You can open the generated Java class once the code generation is completed.

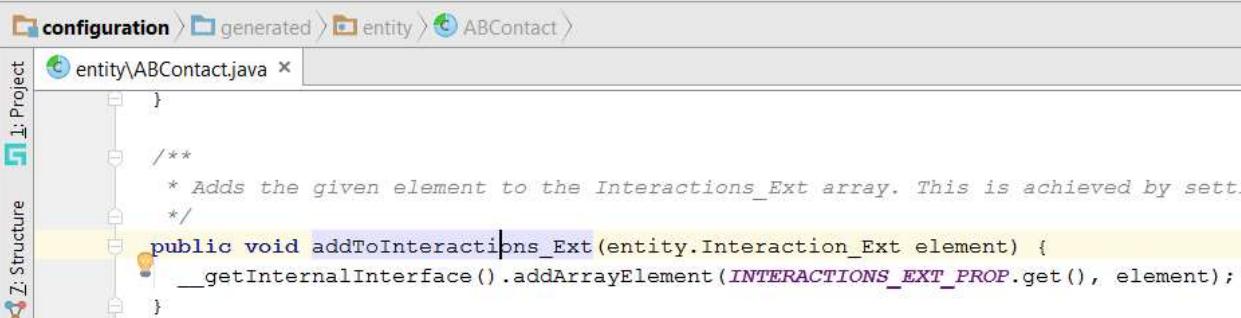
**Note:** The code generation was successful if the Codegen tool window has nothing to show.



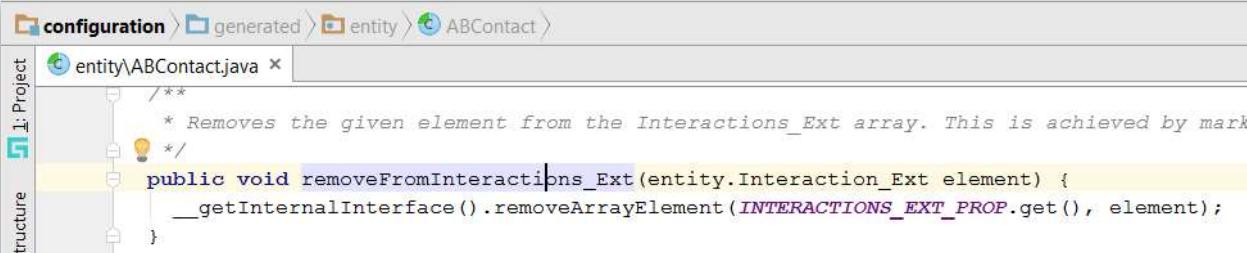
```
configuration > generated > entity > Interaction_Ext
Project: Interaction_Ext.java
public class Interaction_Ext extends com.guidewire.code.BeanBase implements entity.Retireable {
  public static final TYPE = new com.guidewire.commons.metadata.types.EntityIntrinsicTypeReference<~>("entity.Interaction_Ext");
  public static final ILinkPropertyInfo ABCCONTACT_PROP = new com.guidewire.commons.metadata.types.LinkPropertyInfoCache(TYPE, "ABContact");
}
```



```
configuration > generated > entity > ABContact
Project: entity\ABContact.java
public class ABContact extends com.guidewire.code.BeanBase implements entity.Retireable {
  public static final IColumnPropertyInfo ID_PROP = new com.guidewire.commons.metadata.types.ColumnPropertyInfoCache(TYPE, "ID");
  public static final IArrayPropertyInfo INTERACTIONS_EXT_PROP = new com.guidewire.commons.metadata.types.ArrayPropertyInfoCache(TYPE, "Interactions_Ext");
}
```



```
configuration > generated > entity > ABContact
Project: entity\ABContact.java
  }
}
/**
 * Adds the given element to the Interactions_Ext array. This is achieved by setting
 * the element to the array's last position.
 */
public void addToInteractions_Ext(entity.Interaction_Ext element) {
  __getInternalInterface().addArrayElement(INTERACTIONS_EXT_PROP.get(), element);
}
```



The screenshot shows a Java code editor with the following details:

- Project Structure:** The left sidebar shows a tree view of the project structure under "Project 1".
- File Path:** The top navigation bar shows the path: configuration > generated > entity > ABContact.
- Code Editor:** The main area displays the ABContact.java file content.

```
entity\ABContact.java x
/*
 * Removes the given element from the Interactions_Ext array. This is achieved by mark
 */
public void removeFromInteractions_Ext(entity.Interaction_Ext element) {
    __getInternalInterface().removeArrayElement(INTERACTIONS_EXT_PROP.get(), element);
}
```

## Lesson 2 List Views

The business analysts want to capture specific information about each contact interaction such as a telephone call, email message, postal mail letter, or in-person office visit. Eventually, they want the information displayed in an Interactions section of TrainingApp.

| Date                                | Summary  | Associated User |
|-------------------------------------|--|-----------------|
| <input type="checkbox"/> 05/25/2016 | Customer called to make a complaint            | Alice Applegate |
| <input type="checkbox"/> 06/01/2016 | Customer requested proof of coverage statement | Bruce Baker     |

*"We want a CSR (Customer Service Representative) such as Alice Applegate to be able to create and edit details about contact interactions."* – Insurance company business analysts

In the previous lab, you created the Interaction\_Ext entity and modified the ABContact entity to capture interactions. In this lab, you will implement the necessary **user interface changes** to display the interactions. You will also make the ListView editable in a later lab. As a configuration developer, you will use the PCF Editor in Guidewire Studio to modify the TrainingApp user interface.

### 2.1 Prerequisites

You must first complete the following previous lesson(s):

- Creating New Entities

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.

<http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 2.2 Lab: Create a List View

As a configuration developer, you want to be able to create reusable list view panels to be able to display information about multiple entities. In this lab, you will use the PCF Editor to create a reusable List View.

**1. Create and configure the InteractionsLV PCF file:**

- a) Create a List View Panel in the `traininglabs` folder
- b) Specify a root object of the type `ABContact`
- c) Each row of the List View must display the following:
  - `InteractionDate` field labeled as `Date`
  - `Summary` field labeled as `Summary`
  - `AssociatedUser` field labeled as `User` (use the `AltUserCell` widget)

**2. Use Studio's code generation feature to process the PCF file and generate all the resources**

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Verify that you can see generated resources (`.gs`, `.pcfc`, `Expressions.gs`)



### Hint

Configuring cells

Remember, you can always use the Data Dictionary, the generated Java class, and the widget reference table to find out the name, database type and Gosu type of a field that needs to be displayed. Cells are atomic widgets and similar to input widgets.



### Hint

Helpful hints that help you to completion.

The required variables tab defines the input parameters for a container, while the variables tab defines local variables to store values temporarily. (E.g. storing the result of an expensive function call to use it at multiple places in the PCF)

## 2.3 Lab: Reference the List View

As a configuration developer, you want to be able to reference reusable containers. In this lab, you will add the InteractionsLV to the ABContactInteractionsPage PCF file.

### 1. Navigate to the ABContactInteractionsPage and replace the inline Detail View

- a) Delete the Detail View
- b) Add a reference to the InteractionsLV
- c) Add a Toolbar to support paging



#### Hint

Toolbar

Every List View needs to have a Toolbar associated with it. Don't forget that List Views support pagination and the buttons will automatically appear when the number of elements exceeds the pageSize.

### 2.3.1 Verification



#### Activity

Verify the work you have done

1. Log in to TrainingApp as Alice Applegate
2. Use the appropriate shortcut to deploy the changes
3. View the Interactions page for William Andy
  - a) Navigate to William Andy
  - b) In the sidebar menu, click Interactions

- c) Verify that you see the Interactions List View (Note: There will be no data in the List View for now. This is because there are no interactions in the database yet. You will be able to insert new interactions in the Editable List Views lesson.)

The screenshot shows a software interface for managing interactions. On the left, a sidebar menu lists several categories: Actions, Summary, Details, Addresses (3), Notes (5), Social Media, Analysis, and Interactions. The 'Interactions' item is highlighted with a blue background. The main content area has a header 'Person: William Andy' and a title 'Interactions'. Below the title is a toolbar with an 'Edit' button and a dropdown menu icon. A table header row contains columns for 'Interaction Date', 'Summary', and 'Associated User'. A message 'No data to display' is centered below the table. The overall theme is dark with light-colored UI elements.

## 2.4 Bonus lab: List Views and navigation

*"In the screenshot below, the list view panel shows all the employee contacts associated with the employer. Currently, there is no navigable link in the list view panel to the employee contact. A user would need to search for the specific employee contact to see more details. Modify the list view panel so that each employee name links to the contact details page. For example, clicking on William Andy in the table would take us to the Details page of William Andy." – Insurance company business analysts*

| Name             | Job Title | Email Address           |
|------------------|-----------|-------------------------|
| James Andersen   | Attorney  | jandersen@legal.com     |
| Samantha Andrews | Paralegal | sandrews@andrewsmid.com |
| Eric Andy        | Associate | ericandy@albertsons.com |
| William Andy     | Manager   | willandy@albertsons.com |



### Hint

Navigation

If needed, review the **Introduction to Locations** lesson.

### 2.4.1 Verification

1. Log in to TrainingApp as Alice Applegate
2. Reload the PCF changes

### 3. Go to the Details page for Albertson's

- a) Confirm that you are able to click a navigable link for each employee contact

Company: Albertson's

### Details

| Actions       |  | Company Info              |                                     |       | Phone & Addresses |  |                  | Bank Accounts           |                         |               |
|---------------|--|---------------------------|-------------------------------------|-------|-------------------|--|------------------|-------------------------|-------------------------|---------------|
| Summary       |  |                           |                                     |       |                   |  |                  |                         |                         |               |
| Details       |  |                           |                                     |       |                   |  |                  |                         |                         |               |
| Addresses (1) |  | Name                      | Albertson's                         |       |                   |  |                  | Employee Info           |                         |               |
| Notes (0)     |  | Primary Contact           | William Andy                        |       |                   |  |                  | Can Have Employees?     | Yes                     |               |
| Social Media  |  | Address                   | 345 Fir Lane<br>La Canada, CA 91352 |       |                   |  |                  | Number of Employees     | 4                       |               |
| Analysis      |  | Email Address             | willandy@albertsons.com             |       |                   |  |                  | Employees               | Cascade Email Address   |               |
| Interactions  |  | Prefers Contact By Email? | No                                  |       |                   |  |                  | Name                    | Job Title               | Email Address |
| History       |  | Additional Info           |                                     |       |                   |  | James Andersen   | jandersen@elegal.com    |                         |               |
|               |  | Tax ID                    |                                     | ***** |                   |  | Samantha Andrews | sandrews@andrewsmd.com  |                         |               |
|               |  |                           |                                     |       |                   |  | Eric Andy        | ericandy@albertsons.com |                         |               |
|               |  |                           |                                     |       |                   |  | William Andy     | Manager                 | willandy@albertsons.com |               |

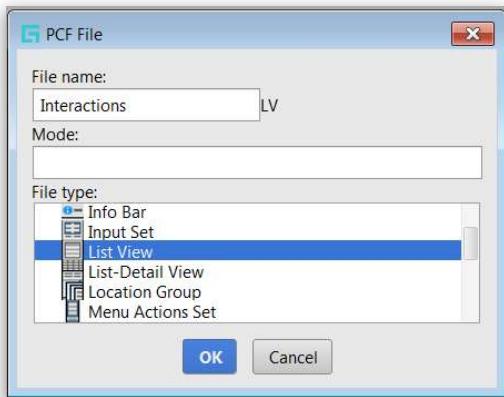


## 2.5 Lab Solution: Create a List View



Exact details on how to complete the lab.

1. Create traininglabs PCF folder if it doesn't exist.
2. Create the List View.



3. Specify the root object.

A screenshot of the Guidewire PCF Editor interface. The title bar says "InteractionsLV.pcf". The main area shows a red-highlighted "List View" component. Below the editor are tabs: "Page Configuration" (selected) and "Text". A toolbar with icons for adding, deleting, and reordering components is visible. The "Properties" tab is active, showing a "Require" section with two entries:

|             |              |             |
|-------------|--------------|-------------|
| anABContact | <b>name*</b> | anABContact |
|             | <b>type*</b> | ABContact   |

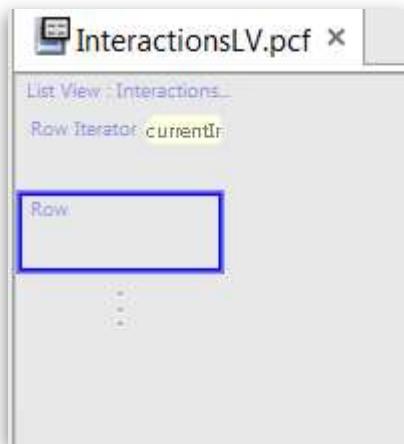
#### 4. Add and configure the Row Iterator

The screenshot shows the Guidewire Studio interface with the following details:

- Title Bar:** The window title is "InteractionsLV.pcf".
- Left Panel:** Shows a tree view with "List View : Interactions..." expanded, and "Row Iterators" selected.
- Toolbar:** Includes tabs for "Page Configuration" (selected) and "Text". Below the tabs are buttons for "Properties", "Variables", "Toolbar Flags", and "Sorting".
- Content Area:** The main area is titled "Row Iterator". A section titled "Basic properties" is expanded, showing the following configuration:

|                     |                               |
|---------------------|-------------------------------|
| <u>editable*</u>    | true                          |
| <b>elementName*</b> | currentInteraction            |
| id                  |                               |
| toAdd               |                               |
| toCreateAndAdd      |                               |
| toRemove            |                               |
| <u>value*</u>       | anABCContact.Interactions_Ext |
| <b>valueType*</b>   | Interaction_Ext[]             |

**5. Add the Row**



## 6. Add the Date Cell

The screenshot shows the Guidewire Studio interface with the following details:

- Title Bar:** InteractionsLV.pcf
- List View:** Interactions...
- Row Iterator:** currentInteraction
- Content Area:** A date input field labeled "Date" is selected. A blue box highlights the "Date" button and the date input field itself.
- Page Configuration Tab:** Active tab is "Text". Other tabs include "Page Configuration" and "Properties".
- Properties Panel:**
  - Date Cell** section is expanded.
  - Basic properties:**

|            |                                       |
|------------|---------------------------------------|
| action     |                                       |
| dateFormat | <none selected>                       |
| editable   | true                                  |
| <b>id*</b> | InteractionDate                       |
| label      | DisplayKey.get("Ext.InteractionDate") |
| required   |                                       |
| timeFormat | <none selected>                       |
| value      | currentInteraction.InteractionDate    |

## 7. Add the Text Cell

The screenshot shows the 'InteractionsLV.pcf' page configuration editor. At the top, there's a toolbar with icons for 'List View', 'InteractionsLV', and 'anABContact'. Below the toolbar, the 'Row Iterator' is set to 'currentInteraction'. The main area displays a row structure with two columns: 'Date' and 'Summary'. The 'Summary' column contains a date input field and a text input field labeled 'currentInter...'. The text input field is highlighted with a blue border. Below the row, there are two empty rows. The bottom section of the editor shows the 'Text' tab selected in the 'Page Configuration' tab bar. The 'Properties' tab is active, displaying the following configuration for a 'Text Cell':

|                 |                               |
|-----------------|-------------------------------|
| <u>action</u>   |                               |
| <u>editable</u> | true                          |
| <u>id*</u>      | InteractionSummary            |
| <u>label</u>    | DisplayKey.get("Ext.Summary") |
| <u>required</u> |                               |
| <u>value</u>    | x currentInteraction.Summary  |

## 8. Add the AltUserCell

The screenshot shows the 'InteractionsLV.pcf' page configuration interface. At the top, it displays the page title 'List View : InteractionsLV \ anABContact' and the row iterator 'currentInteraction'. Below this, there is a grid structure with three columns labeled 'Date', 'Summary', and 'User'. The 'User' column contains a placeholder 'AltUser...'. A blue box highlights this placeholder. Below the grid, there are sections for 'Page Configuration' and 'Text'. Under 'Properties', the 'Properties' tab is selected. In the properties panel, under the 'Basic properties' section, the 'value' field for the 'AltUserCell' is set to 'currentInteraction.AssociatedUser'. The 'label' field is set to 'DisplayKey.get("Ext.AssociatedUser")'. Other properties like 'action', 'editable', and 'required' are also listed.

| Property     | Value                                |
|--------------|--------------------------------------|
| action       |                                      |
| editable     | true                                 |
| <b>id*</b>   | AssociatedUser                       |
| label        | DisplayKey.get("Ext.AssociatedUser") |
| required     |                                      |
| <b>value</b> | currentInteraction.AssociatedUser    |

## 2.6 Lab Solution: Reference the List View



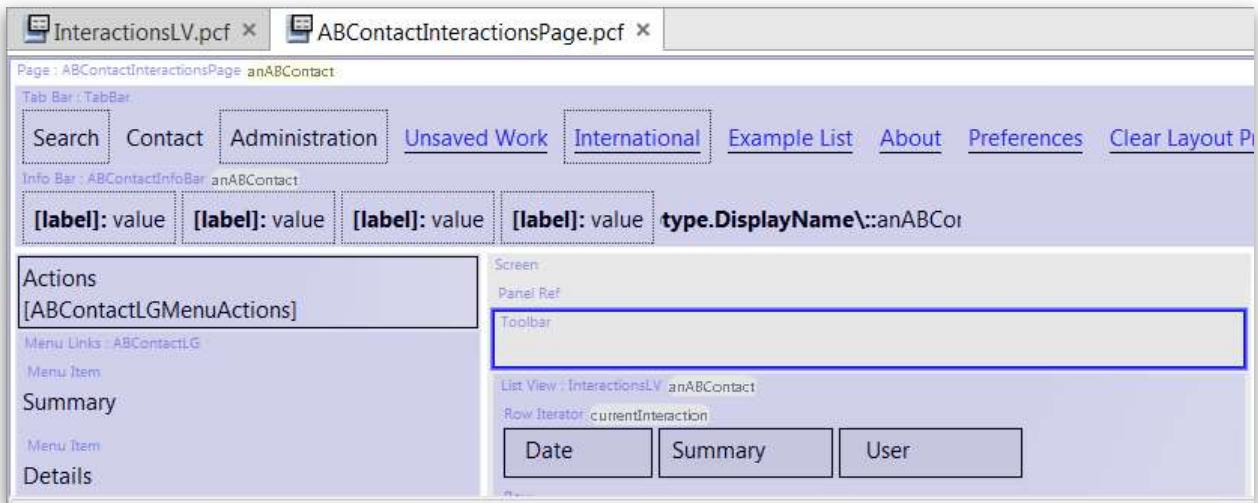
### Solution

Exact details on how to complete the lab.

1. Open the ABContactInteractionsPage and replace the DetailView with a PanelRef.

The screenshot shows the configuration interface for the ABContactInteractionsPage.pcf page. The top bar displays two tabs: 'InteractionsLV.pcf' and 'ABContactInteractionsPage.pcf'. The main area is titled 'Page : ABContactInteractionsPage anABContact'. A 'Tab Bar' contains tabs like Search, Contact, Administration, and International. Below it is an 'Info Bar' with an 'Actions' section containing '[ABContactLGMenuActions]'. The 'Screen' panel on the right shows a 'Panel Ref' section with a 'List View' component. The 'Basic properties' table in the bottom right has a row where 'def' is set to 'InteractionsLV (anABContact)'.

## 2. Add a Toolbar into the PanelRef



## 2.7 Bonus Lab Solution: List Views and navigation



### Solution

Exact details on how to complete the lab.

1. Define the **action** property of the **TextCell** in **ABCompanyEmployeesLV**

The screenshot shows the 'ABCompanyEmployeesLV.pcf' page configuration window. At the top, it displays the page title 'List View : ABCompanyEmployeesLV anABContact' and the row iterator 'Row Iterator currentEmployee'. Below this, there is a grid of three columns labeled 'Name', 'Job Title', and 'Email Address'. The first row of the grid contains three text cells, each with the value 'currentEmpl...'. The second row is partially visible. In the bottom right corner of the first cell of the first row, there is a small blue rectangular button with a white outline. The 'Properties' tab is selected in the bottom navigation bar. In the 'Text Cell' section of the properties panel, under the 'Basic properties' group, the 'action' property is set to 'ABPersonDetailsPage.go(currentEmployee)'. Other tabs like 'Page Configuration' and 'Text' are also visible.

## Lesson 3 Editable List Views

| Date       | Summary                               | Associated User |
|------------|---------------------------------------|-----------------|
| 05/25/2016 | Customer called to file a complaint   | Alice Applegate |
| 06/01/2016 | (Customer called to file a complaint) | Bruce Baker     |

(1) Add Edit Buttons, so that users can edit the Interactions page  
(2) Make the List View editable, so that users can manually add and remove interactions  
(3) Checkboxes should be visible only if the List View is in edit mode

*"We want a CSR (Customer Service Representative) such as Alice Applegate to be able to create and edit details about contact interactions."* – Insurance company business analysts

In the previous labs, you created the Interaction\_Ext entity, modified the ABContact entity to capture interactions and created a reusable read-only List View to display Interactions. In this lab, you will add Add and Remove buttons to ABContactInteractionsPage to allow the user to add and remove interactions.

### 3.1 Prerequisites

You must first complete the following previous lesson(s):

1. Creating New Entities
2. List Views

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.

<http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 3.2 Lab: Make the Interactions List View editable

As a configuration developer, we want to be able to configure the Iterator Buttons widget to allow users to add and remove rows. In this lab, you will use the PCF Editor in Guidewire Studio to make the Interactions List View editable.

### 1. Modify the Interactions page so that users can add and remove interactions

- a) The page must have a toolbar with Edit/Update/Cancel buttons and with Add/Remove buttons
- b) Users must be able to add and remove rows upon selecting the appropriate buttons
- c) Ensure that data in all three columns are editable
- d) Show the checkboxes by each row only when in edit mode

### 3.2.1 Verification



#### Activity

Verify the work you have done

1. Log in to TrainingApp as Alice Applegate
2. Use the appropriate shortcut to deploy the changes
3. View the Interactions page for William Andy
  - a) Navigate to William Andy
  - b) In the sidebar menu, click Interactions
  - c) Click Edit and add at least three interactions

d) Click Update

The screenshot shows a software application window for managing customer interactions. On the left, a sidebar menu lists various actions: Actions, Summary, Details, Addresses (3), Notes (5), Social Media, Analysis, and Interactions. The 'Interactions' option is selected and highlighted with a blue bar at the bottom. The main content area is titled 'Interactions' and displays a table of recent interactions for a person named 'Person: William Andy'. The table has columns for 'Interaction Date', 'Summary', and 'Associated User'. The data in the table is as follows:

| Interaction Date | Summary                              | Associated User |
|------------------|--------------------------------------|-----------------|
| 09/24/2018       | Quarterly courtesy contact           | Alice Applegate |
| 04/25/2018       | Customer requested change in address | Carl Clark      |
| 06/22/2018       | Quarterly courtesy contact           | Bruce Baker     |



### 3.3 Lab Solution: Make the Interactions List View editable



#### Solution

Exact details on how to complete the lab.

1. Open ABContactInteractionsPage and add Edit Buttons and Iterator Buttons widgets. Note: The Iterator Buttons widget will be red because the `iterator*` property is required

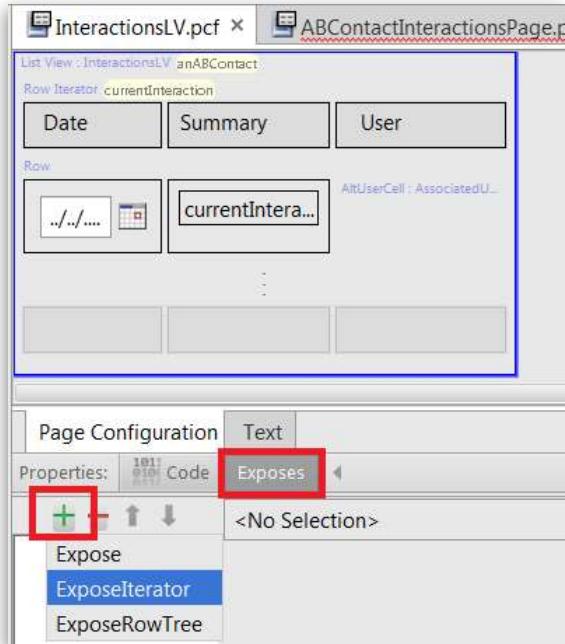
The screenshot shows the 'ABContactInteractionsPage.pcf' page configuration. The main content area displays a grid of four columns with labels: '[label]: value'. Below this grid is a section titled 'Actions' containing '[ABContactLGMenuActions]'. To the right of the actions are several buttons: 'Edit' (green), 'Update' (green), 'Cancel' (green), 'Add' (red), and 'Remove' (red). A tooltip for the 'Add' button indicates 'List View : InteractionsLV::anABContact'. The 'Row Iterator' is set to 'currentInteraction'. At the bottom of the page configuration, there are tabs for 'Page Configuration' and 'Text', and a 'Properties' panel.

**Properties Panel:**

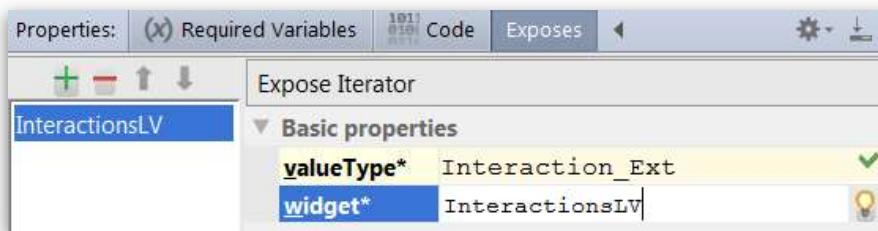
| Iterator Buttons        |
|-------------------------|
| <b>Basic properties</b> |
| <code>addVisible</code> |
| <code>id</code>         |
| <code>iterator*</code>  |

## 2. Expose the Row Iterator

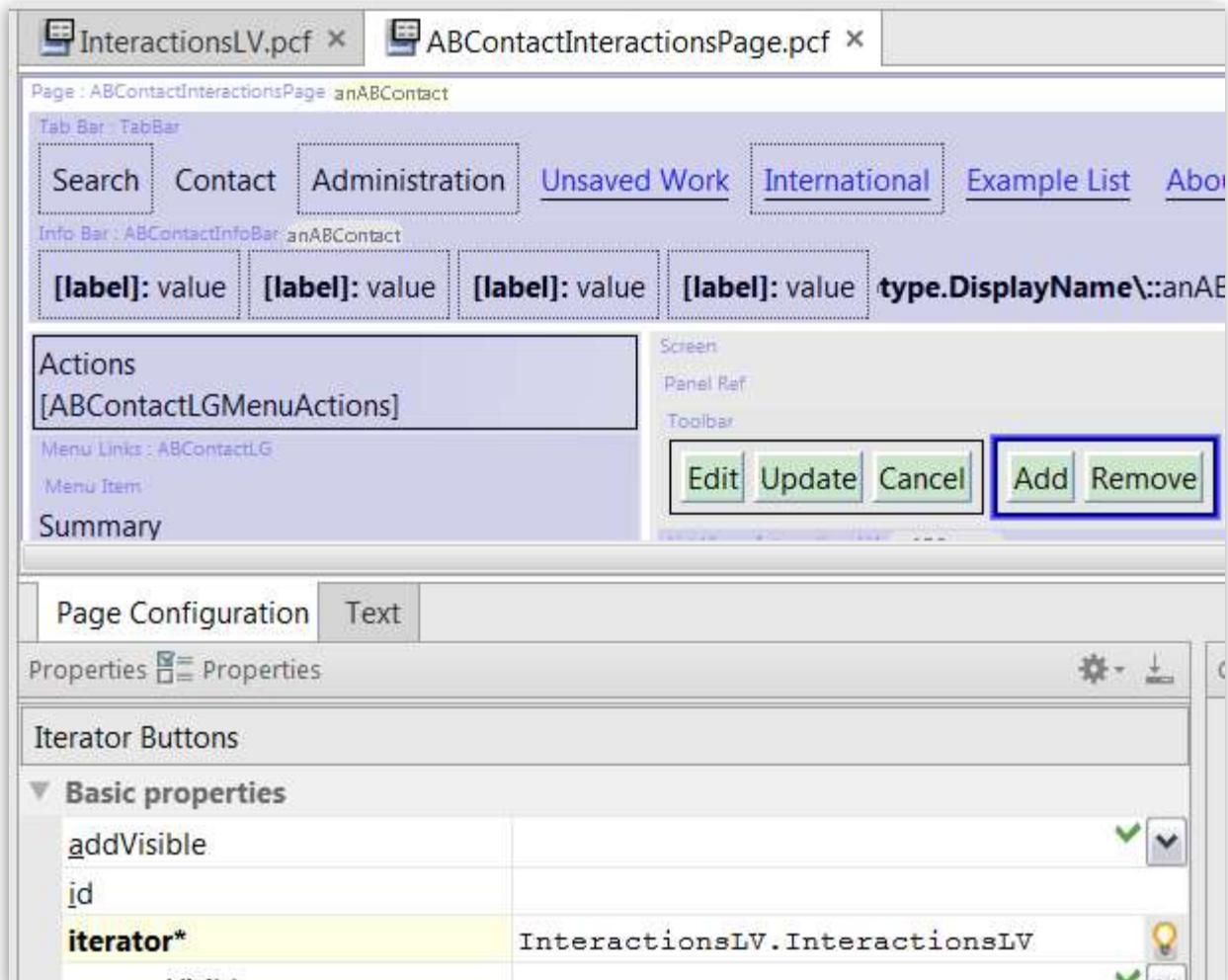
- a) Open the InteractionsLV, go to the Exposes tab and click on the green plus icon to expose the iterator



- b) Configure the **valueType** and **widget** properties of the Expose Iterator



3. Reference the new Expose Iterator from the iterator property of the Iterator Buttons

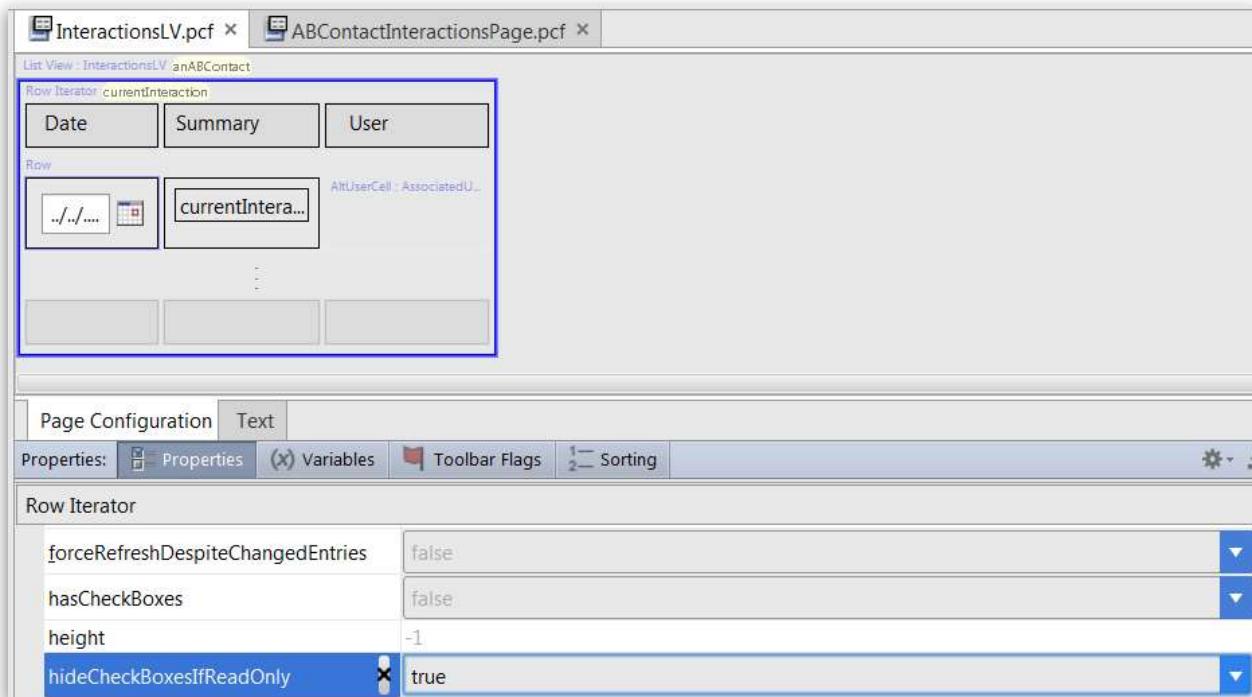


#### 4. Configure the toAdd and toRemove properties of the Row Iterator

The screenshot shows the configuration of a Row Iterator named 'currentInteraction' within a List View component. The Row Iterator is defined with three columns: Date, Summary, and User. The 'toAdd' property is set to a JavaScript function that adds the current interaction to the contact's interactions list. The 'toRemove' property is set to a JavaScript function that removes the current interaction from the contact's interactions list.

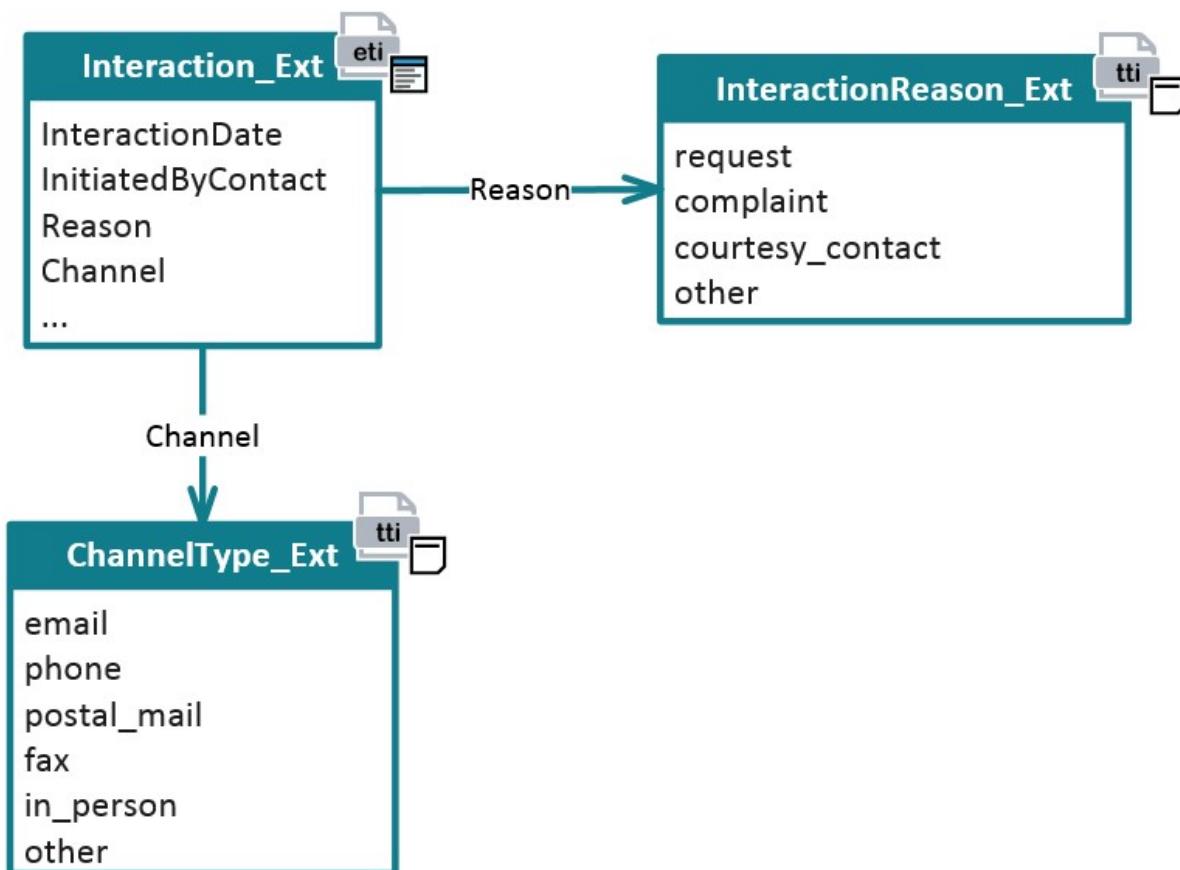
| Property       | Value  |
|----------------|--|
| editable*      | true   |
| elementName*   | currentInteraction   |
| id             |  |
| toAdd          | anABContact.addToInteractions_Ext(currentInteraction)      |
| toCreateAndAdd |  |
| toRemove       | anABContact.removeFromInteractions_Ext(currentInteraction) |
| value*         | anABContact.Interactions_Ext                               |
| valueType*     | Interaction_Ext[]  |

5. Set the **hideCheckboxesIfReadOnly** property to true



## Lesson 4 Typelists

*"After further analysis, we decided that we want to store more information about each contact interaction. Please see the diagrams below and configure the Data Model to meet the requirements." – Insurance company business analysts*



In this lab, you will use the Typelist Editor and the Entity Editor in Guidewire Studio to modify the TrainingApp data model. First, you will create two new typelists. Then, you will edit existing entities and associate the new typelists with each respective entity.

## 4.1 Prerequisites

You must first complete the following previous lesson(s):

### 6. Creating New Entities

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.

<http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 4.2 Configuration

### 4.3 Lab: Create new typelists

As a configuration developer, you want to be able to create new typelists from scratch. In this lab, you will create and define two typelists in Guidewire Studio using the Typelist Editor. The InteractionReason\_Ext typelist will describe the reasons for contact interactions. The ChannelType\_Ext typelist will describe the various channels of communications for interactions.

#### 1. Create the InteractionReason\_Ext typelist

- a) Create the new TTI file
- b) Add the following typecodes to display in the order listed using recommended naming conventions:
  - Request
  - Complaint
  - Courtesy Contact
  - Other

#### 2. Use Studio's code generation feature to process the typelist and generate the Java class

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Open the generated Java class and verify that you can see the newly added typecodes

### 3. Create the ChannelType\_Ext typelist

- a) Create the new TTI file
- b) Add the following typecodes to display in the order listed using the recommended naming conventions:
  - Email
  - Phone
  - Postal Mail
  - Fax
  - In Person
  - Other

### 4. Use Studio's code generation feature to process the typelist and generate the Java class

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Open the generated Java class and verify that you can see the newly added typecodes



#### Best Practices

Use \_Ext in the typelist name

Notice that the name of the typelist has an \_Ext suffix. For new typelists, Guidewire recommends that the name of the typelist should end with \_Ext (or start with Ext\_). This is to prevent potential conflicts during the upgrade to the next version of the Guidewire application.

## 4.4 Lab: Add typekey fields to the entity

As a configuration developer, you want to be able to add typekey entity fields that reference typelists. In this lab, you will edit the related entities and add new typekey elements for the newly created typelists.

### 1. Edit the Interaction\_Ext entity

- a) Add a typekey field for the InteractionReason\_Ext typelist using the recommended naming convention
- b) Add a typekey field for the ChannelType\_Ext using the recommended naming convention

### 2. Use Studio's code generation feature to generate the Java class from the entity

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Open the generated Java class and verify that you can see the two new typekey fields



## Important!

Read carefully.

For each new entity element, remember to set the `nullOk` attribute to true, otherwise, the database upgrade may fail.



## Best Practices

Use `_Ext` in the entity field name

Notice that the name of the typelist has an `_Ext` suffix. For new typelists, Guidewire recommends that the name of the typelist should end with `_Ext` (or start with `Ext_`). This is to prevent potential conflicts during the upgrade to the next version of the Guidewire application.

### 4.4.1 Verification



## Activity

Verify the work you have done

As a configuration developer, you want to be able to properly deploy new and changed data model resources.

#### 1. Restart the server to deploy the changes

- a) During server restart Studio first runs the code generators, then compiles the project and finally deploys the resources

#### 2. Regenerate the Data Dictionary

- a) Follow the steps to regenerate the Data Dictionary

#### 3. Open the Data Dictionary

- a) In Windows Explorer, navigate to the data dictionary.
  - b) Open the data dictionary using your preferred browser.
4. Verify the InteractionReason\_Ext and ChannelType\_Ext typelists
  5. Verify the new typelist reference in the Interaction\_Ext entity



## 4.5 Lab Solution: Create new typelists



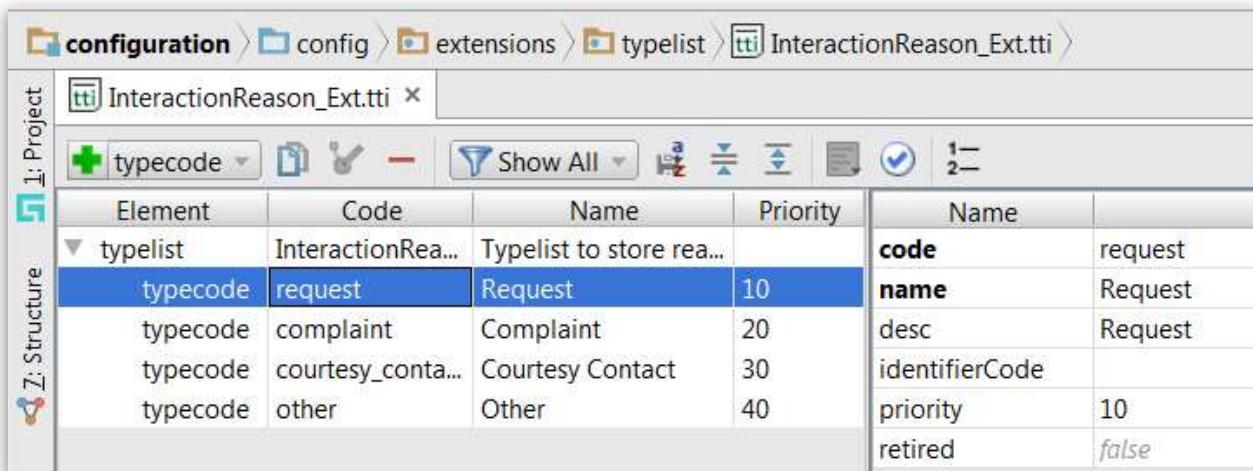
### Solution

Exact details on how to complete the lab.

#### 1. Create the new InteractionReason\_Ext typelist

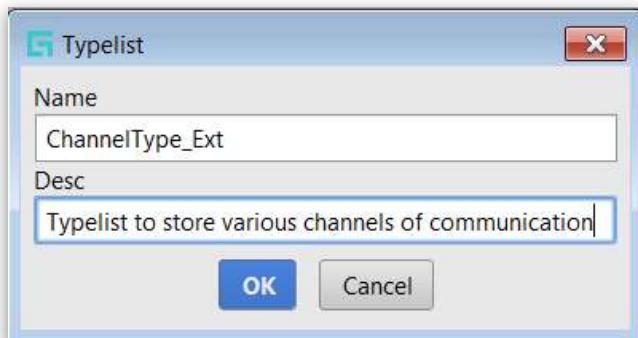


## 2. Add the typecodes



| Element  | Code              | Name                     | Priority | Name           |         |
|----------|-------------------|--------------------------|----------|----------------|---------|
| typelist | InteractionRea... | Typelist to store rea... |          | code           | request |
| typecode | request           | Request                  | 10       | name           | Request |
| typecode | complaint         | Complaint                | 20       | desc           | Request |
| typecode | courtesy_contact  | Courtesy Contact         | 30       | identifierCode |         |
| typecode | other             | Other                    | 40       | priority       | 10      |
|          |                   |                          |          | retired        | false   |

## 3. Create the new ChannelType\_Ext typelist



#### 4. Add typecodes

The screenshot shows the Guidewire configuration interface with the following path in the breadcrumb: configuration > config > extensions > typelist > ChannelType\_Ext.tti. The left sidebar has two sections: 1: Project and 2: Structure. The main area displays a table titled 'ChannelType\_Ext.tti' with the following data:

| Element  | Code                | Name               | Priority | Name           |       |
|----------|---------------------|--------------------|----------|----------------|-------|
| typelist | ChannelType_Ext.tti | Typelist to sto... |          | code           | email |
| typecode | email               | Email              | 10       | name           | Email |
| typecode | phone               | Phone              | 20       | desc           | Email |
| typecode | postal_mail         | Postal Mail        | 30       | identifierCode |       |
| typecode | fax                 | Fax                | 40       | priority       | 10    |
| typecode | inperson            | In Person          | 50       | retired        | false |
| typecode | other               | Other              | 60       |                |       |

## 4.6 Lab Solution: Add typekey fields to an entity



### Solution

Exact details on how to complete the lab.

1. Open Interaction\_Ext.eti
2. Add a typekey field that uses the new InteractionReason\_Ext typelist

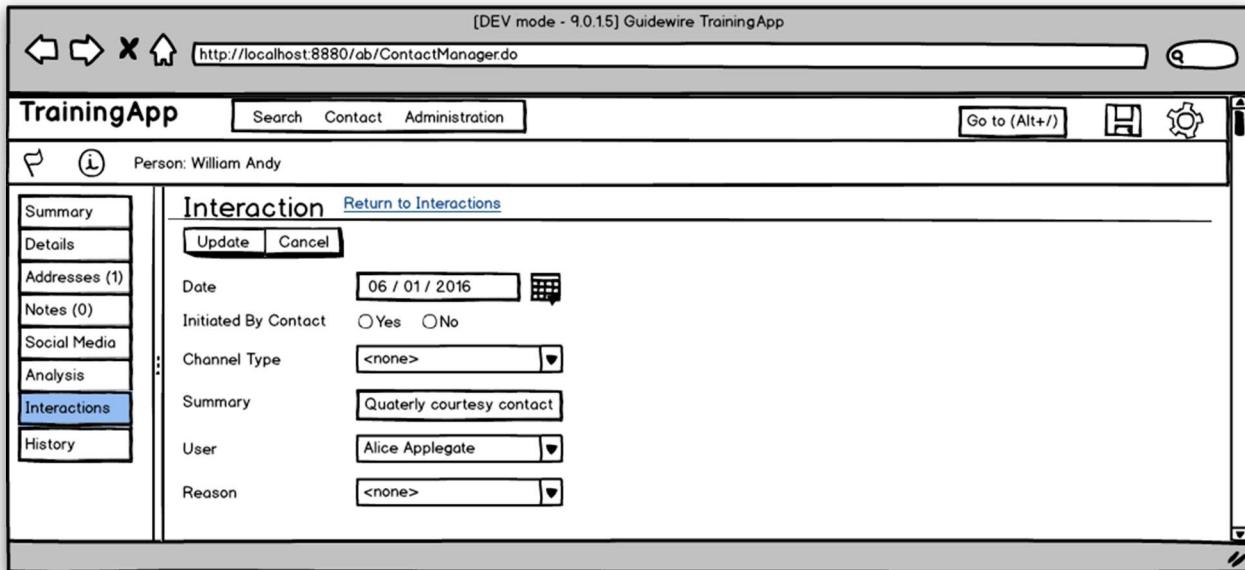
| Element        | Primary Value      | Secondary Value            | Name                               |                       |
|----------------|--------------------|----------------------------|------------------------------------|-----------------------|
| entity         | Interaction_Ext    | Stores interactio...       | <b>name</b>                        | Reason                |
| column         | InteractionDate    | datetime                   | <b>typelist</b>                    | InteractionReason_Ext |
| column         | InitiatedByContact | bit                        | <b>nullok</b>                      | true                  |
| ▶ column       | Summary            | varchar                    | <b>desc</b>                        |                       |
| foreignkey     | AssociatedUser     | User                       | <b>allowInitialValueForUpgrade</b> | <i>false</i>          |
| foreignkey     | ABContact          | ABContact                  | <b>columnName</b>                  |                       |
| <b>typekey</b> | <b>Reason</b>      | <b>InteractionReaso...</b> | <b>createhistogram</b>             | <i>false</i>          |

3. Add a typekey field that uses the new ChannelType\_Ext typelist

| Element    | Primary Value      | Secondary Value      | Name                              |
|------------|--------------------|----------------------|-----------------------------------|
| entity     | Interaction_Ext    | Stores interactio... | <b>name</b> Channel               |
| column     | InteractionDate    | datetime             | <b>typelist</b> ChannelType_Ext   |
| column     | InitiatedByContact | bit                  | <b>nullok</b> true                |
| ▶ column   | Summary            | varchar              | desc                              |
| foreignkey | AssociatedUser     | User                 | allowInitialValueForUpgrade false |
| foreignkey | ABContact          | ABContact            | columnName                        |
| typekey    | Reason             | InteractionReaso...  | createhistogram false             |
| typekey    | Channel            | ChannelType_Ext      | default                           |

## Lesson 5 Popups: View and Edit

*“Currently, users can add, remove, and edit contact interactions on the Interactions page in the Interactions list view panel. However, the editable list view panel on the page only exposes some of the Interaction entity fields. Configure a popup that will allow users to view and edit all the fields of a contact interaction.” – Insurance company business analysts*



As a configuration developer, you want to be able to create, configure and open popup locations so that the users can see and/or edit more information about an object. In this lab, you will first create the Interaction Popup. The popup will allow users to edit and view an interaction. Next, you will modify the InteractionsLV. You will configure a cell widget in the list view panel to navigate to the popup.

### 5.1 Prerequisites

You must first complete the following previous lesson(s):

1. Creating New Entities
2. List Views
3. Editable List Views
4. Typelists

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.  
`http://localhost:8880/ab/ContactManager.do` is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is `aapplegate/gw`.

## 5.2 Lab: Popup locations

### 1. In Project View, create the InteractionPopup in the traininglabs PCF folder

- a) Create the traininglabs PCF folder if it doesn't exist
- b) For the Popup, set the `startEditMode` and `canEdit` properties to true
- c) To the Popup, add an inline Screen and an inline Detail View
- d) To the inline Detail View, add editable atomic widgets for the following `Interaction_Ext` fields:
  - Interaction Date
  - Initiated By Contact
  - Channel Type
  - Summary
  - Associated User
  - Interaction Reason

### 2. Modify the InteractionsLV.pcf

- a) Modify the cell widget for the Summary field to be a navigable link to the editable Interaction Popup. The Popup should display the entity from the row that the user clicked on.

**Note:** Use your best judgment for field ordering, display keys, and input widget selection. Review the Atomic Widgets lesson for more details if needed.



### Guidewire API

The `startEditMode` property

If this property is set to true then the location will open in edit mode allowing the user to modify the data immediately. If the property is set to false then the location will open in read-only mode and the user will need to click on the edit button to edit the details. This property could be also set dynamically, e.g. based on an expression, result of a function call or a parameter received from the parent location in case of Popups.



## Guidewire API

The `canEdit` property

This property determines if the location (and any containers inside it) can or cannot be in edit mode. If the `startInEditMode` property is set to true or the location has a Toolbar and Edit Buttons widget then the `canEdit` property must be set to true. The default value is false.



### Hint

Referencing the object associated with a row

When you click on a cell in a row, you want to open the popup and pass in the object that is associated with that specific row. Remember, the variable defined in the `elementName` attribute of the Row Iterator is always the variable that will store that associated object. As a configuration developer, you don't have to worry about indexes. You can just simply reference the variable and the PCF framework will determine which object from the array is associated with that row the user clicked on.



### Review

Navigating to a location

Review the Introduction to Locations lesson if you need tips on how to define navigation in Guidewire applications.

## 5.2.1 Verification



### Activity

Verify the work you have done

#### 1. Log in to TrainingApp as Alice Applegate

- a) Use the appropriate shortcut to reload the PCF changes
- b) Edit the Interactions page for William Andy
- c) Open the William Andy contact
- d) In the sidebar menu, click Interactions
- e) If you previously did not create an interaction, create at least one interaction now in the list view panel
- f) Click **Edit** then click **Add**
- g) Enter a date, a reason, and select a user
- h) Click **Update**
- i) For an interaction row, click an interaction reason link
- j) Edit all editable fields in the Interaction Popup
- k) Click **Update**

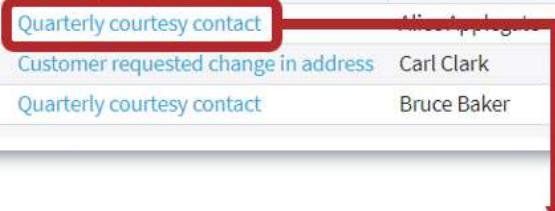
Interactions

| Interaction Date | Summary                              | Associated User |
|------------------|--------------------------------------|-----------------|
| 09/24/2018       | Quarterly courtesy contact           | Mike Appleby    |
| 04/25/2018       | Customer requested change in address | Carl Clark      |
| 06/22/2018       | Quarterly courtesy contact           | Bruce Baker     |

Interaction

Return to Interactions    Update    Cancel

ChannelType: Email



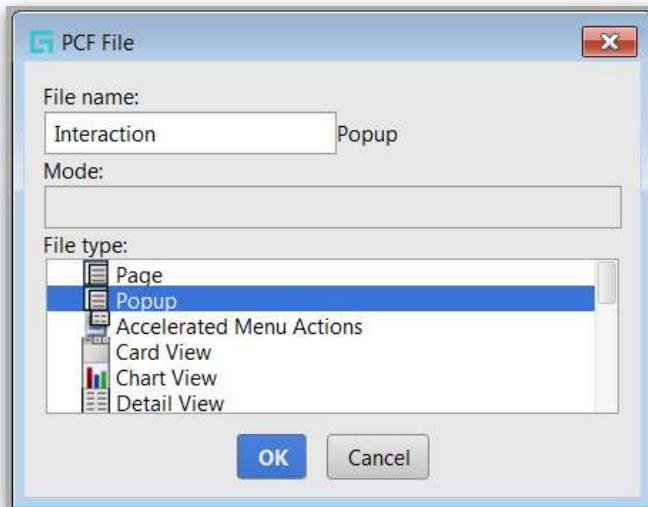
## 5.3 Lab Solution: Popup locations



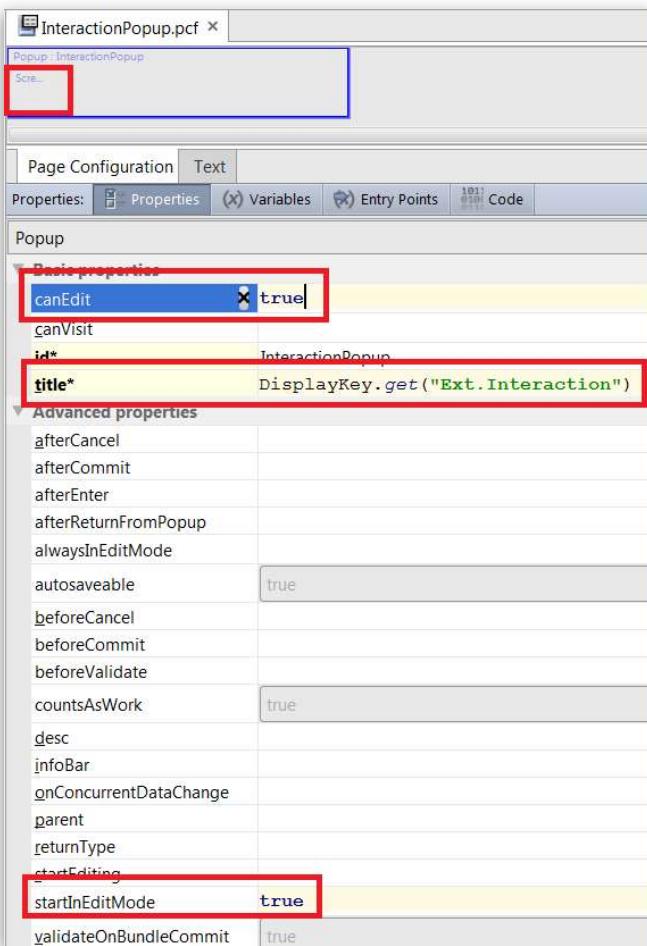
### Solution

Exact details on how to complete the lab.

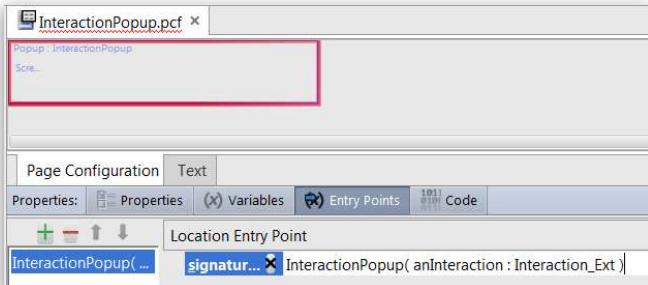
1. Create the traininglabs PCF folder if it doesn't exist
2. Create the InteractionPopup



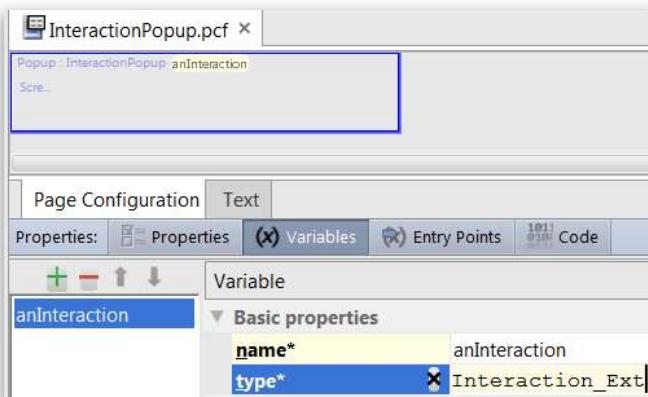
3. Add an inline Screen container and set the following properties: `startInEditMode`, `canEdit`, `title`



#### 4. Specify an entry point with a single Interaction\_Ext parameter



#### 5. Specify a variable for the parameter (same name and same type)



#### 6. Add a toolbar and edit buttons

- From the toolbox, drag and drop a **Toolbar** into the screen you just added.
- Drag and drop an **Edit Buttons** widget into the toolbar.

#### 7. Add a detail view

- Drag and drop a **Detail View** below the **Toolbar** widget
- The widget is colored red because the Detail View must contain an Input Column or a Table Layout. You'll fix this in the next step.

#### 8. Add an Input Column

- Drag and drop an **Input Column** widget into the **Detail View**.
- Notice that the Detail View is no longer red.

**9. Add an input for the Interaction Date field**

- a) Drag and drop a **Date Input** into the **Input Column**.
- b) Set the editable property to **true**.
- c) Set the id property to **InteractionDate**.
- d) Set the label property to **DisplayKey.get("Ext.InteractionDate")**. This display key was created an earlier lab lab. If it does not exist, create it at this point.
- e) Set the value property to **anInteraction.InteractionDate**.

**10. Add an input for the Initiated by Contact field**

- a) Drag and drop a **Boolean Radio Button** Input into the **Input Column** below the Interaction Date input.
- b) Set the editable property to **true**.
- c) Set the id property to **InitiatedByContact**.
- d) Set the label property to **DisplayKey.get("Ext. InitiatedByContact ")**. This is a new display key, so create it using Studio.
- e) Set the value property to **anInteraction.InitiatedByContact**.

**11. Add an input for the Channel Type field**

- a) Drag and drop a **TypeKey Input** into the **Input Column** below the Initiated By Contact input.
- b) Set the editable property to **true**.
- c) Set the id property to **ChannelType**.
- d) Set the label property to **DisplayKey.get("Ext. ChannelType")**.
- e) Set the value property to **anInteraction.Channel**.
- f) Notice that the valueRange and valueType fields were automatically entered.

**12. Add an input for the Summary field**

- a) Drag and drop a **Text Input** into the **Input Column** below the Channel Type input.
- b) Set the editable property to **true**.
- c) Set the id property to **Summary**.
- d) Set the label property to **DisplayKey.get("Ext. Summary")**.
- e) Set the value property to **anInteraction.Summary**.

**13. Add an input for the Associated User field**

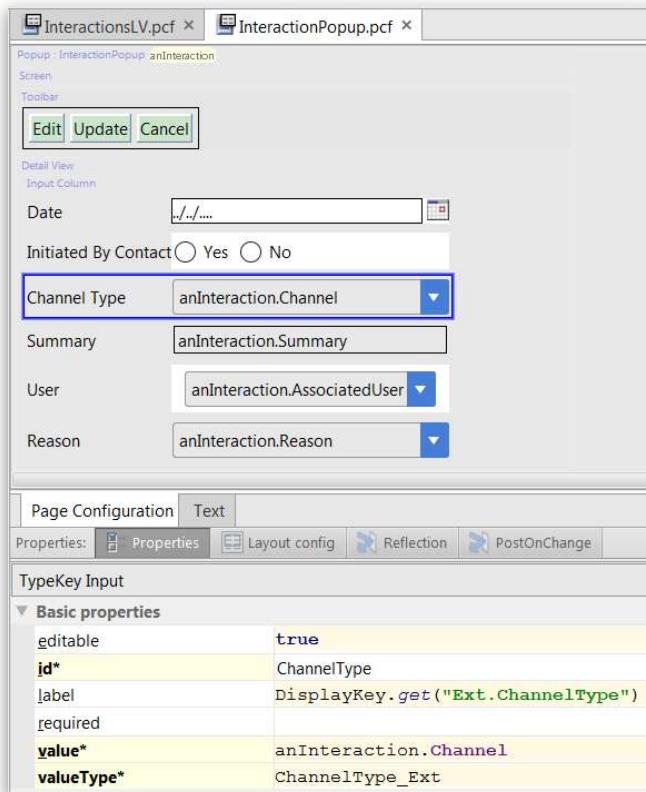
- a) Drag and drop an **Alt User Input** into the **Input Column** below the Summary input.
- b) Set the editable property to **true**.
- c) Set the id property to **AssociatedUser**.

- d) Set the label property to **DisplayKey.get("Ext.AssociatedUser")**.
- e) Set the value property to **anInteraction.AssociatedUser**.

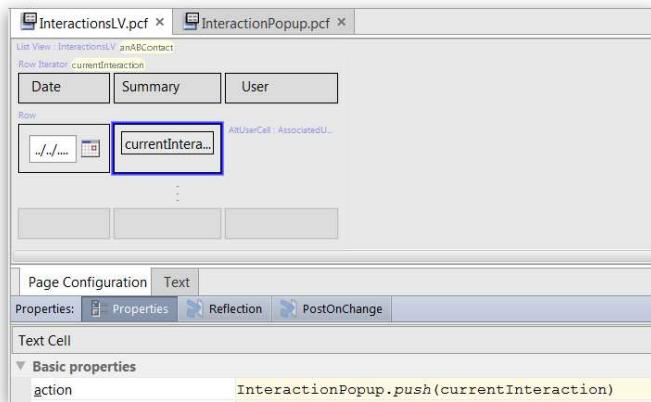
#### 14. Add an input for the Interaction Reason field

- a) Drag and drop a **Range Input** into the **Input Column** below the Associated User input.
- b) Set the editable property to **true**.
- c) Set the id property to **ReasonType**.
- d) Set the label property to **DisplayKey.get("Ext.ReasonType")**.
- e) Set the value property to **anInteraction.Reason**.
- f) Notice that the valueRange and valueType fields were automatically entered.

#### 15. The completed popup should look like this screenshot:



**16. Configure the action property of the Summary cell to open the InteractionPopup**



# Lesson 6 Validation

As a configuration developer, you want to be able to verify the data entered by the user. If the data is incorrect (wrong format, missing information, inconsistencies between fields) then you want to prevent the save. You also want to be able to highlight widgets and show warning/error messages as needed. In this lab, you will make several configuration changes to improve the validation in TrainingApp.

## 6.1 Prerequisites

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.

`http://localhost:8880/ab/ContactManager.do` is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is `aapplegate/gw`.



### Activity

## 6.2 Lab: Field-level validation in the UI

*“Every Auto Repair Shop has a license field that can be specified on the Company Info card. Currently, TrainingApp doesn’t validate this field and allows saving license numbers in any format. Configure TrainingApp to meet the following requirements:*

- Auto Repair Shop licenses must be in the following format: the first character must be a lower-case “a” character, followed by any lower-case alphabet character, then 5 digits. Example: aw-36292
- TrainingApp should display a watermark and tooltip to guide the user
- License numbers cannot contain the string ‘777’. If the license number contains ‘777’ then the save shouldn’t be allowed, and the user should be notified that the license number is invalid.
  - Insurance company business analysts

### 6.2.1 Investigation

#### 1. View the current widget behavior

- a) Log in as Alice Applegate

- b) Search for Burlingame Saab
- c) In the sidebar menu, select Details
- d) Click Edit
- e) Enter any alphanumeric value in the License field
- f) Click Update

## 2. Open the PCF file in Studio

- a) To open the PCF, use ALT + SHIFT + E

### 6.2.2 Configuration

In this part of the lab, you will configure the License Text Input field to show the user a watermark.

#### 1. Modify the License widget to show a watermark

- a) Modify ABContactDetailsCompanyDV.pcf so that the ABAutoRepairShopLicense (ID: ABAutoRepairShopLicense) field shows the following watermark:

a#-#####



In this part of the lab, you will configure the AutoRepairLicense field to match the pattern of the field watermark.

#### 2. Modify the License widget to validate the input so that it matches the watermark

- a) Modify ABContactDetailsCompanyDV.pcf so that the ABAutoRepairShopLicense field matches the pattern of the watermark:
  - The first character must be a lower case "a" alphabet character
  - The second character must be any lower case alphabet character
  - The third character is a dash
  - The fourth, through eighth characters must be numbers

In this part of the lab, you will configure the AutoRepairLicense field to validate the value before it is committed. If the license number contains the string "777", then the user should be notified that license number is invalid.

### 3. Modify the License widget to validate the input based on business logic

- a) If the license number contains the string “777”, then the user should be notified that license number is invalid

#### 6.2.3 Verification

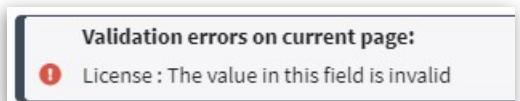


##### Activity

Verify the work you have done

###### 1. Verify the watermark and pattern

- a) In TrainingApp, deploy your changes to the PCF file
- b) Edit the Details for Burlingame Saab
- c) For the License field, enter **ab-12345** and verify that there is no field format warning
- d) Click Update
- e) Click Edit
- f) Enter **a1-12345** and click Update
- g) Verify that there is a validate error message:



###### 2. Verify the validation expression

- a) Change the license field value to ab-12777
- b) Click Update
- c) Verify the info bar error message:



- d) Enter ab-12345 as a valid license field value
- e) Click Update
- f) Verify that you were able to commit the changes



### Hint

An overview of the syntax for validator patterns is shown in the following table. **Note:** this is only a high-level overview. For a complete listing of syntax for validator patterns, consult the Configuration Guide for your product.

|                     |  |
|---------------------|--|
| .                   | Any character  |
| ?                   | Indicates zero or one occurrence of the preceding element  |
| +                   | Indicates one or more occurrences of the preceding element   |
| *                   | Indicates zero or more occurrences of the preceding element  |
| {X}                 | Preceding item is matched exactly X times  |
| {X, Y}              | Preceding item is matched at least X times, but not more than Y times.<br>(technically X is minimum, Y is maximum) |
| .+                  | Any non-empty string (at least one or more characters)   |
| \                   | Escape character   |
| [ ] ( ) { } . * + ? | Characters that are not treated as literals and must be escaped  |

|                    |   |
|--------------------|---|
| [charRange] {X}    | Characters in character range, which can be any combination of 0-9, a-z, and A-Z. X means the exact number of characters we need from the charRange |
| [charRange] {X, Y} | From X to Y characters in charRange   |
| @ - \$             | Most other characters are treated as literals such as a dash (-)  |
| \.                 | A period, where the backslash is the escape character   |

### Examples:

|                                   |  |
|-----------------------------------|--|
| .+@.+                             | Validates an email address. All of these are valid values:<br><a href="mailto:a@b.com">a@b.com</a> <a href="mailto:william@andy.com">william@andy.com</a> <a href="mailto:chris@SUCCEED.com">chris@SUCCEED.com</a> |
| [0-9a-zA-Z] {3} - [0-9] {3}       | This validates the following format: 3 alphanumeric characters, a dash and 3 digits. All of these are valid values: 999-999, AB9-333, A7C-444  |
| [0-9] {3} - [0-9] {2} - [0-9] {4} | This validates the following format: 3 digits, dash, 2 digits, dash and 4 digits. E.g.: 444-55-6666  |

## 6.3 Lab: Field-level validation in the Data Model

*"The insurance company requires that all bank account numbers follow the pattern XX-XXXX, where X is a digit from 0 through 9. The compliance with the pattern must be enforced in the UI as well as through APIs. Without needing to modify existing widget properties in the application, we also require that there is a watermark identifying the pattern. The validation error message for an invalid bank account value must be localizable." – Insurance company business analysts*

### 6.3.1 Configuration

#### 1. Create a validator display key error message

- a) Create a display key in display.properties that reads:

"A bank account number must be two numbers, a hyphen, and then four numbers. For example, 12-3456."

**2. Create a field validator**

- a) Create new ValidatorDef in fieldvalidators.xml
- b) Specify the attributes to meet the lab requirements

**3. Associate the field validator with an entity element**

- a) Add an entity field validator for the AccountNumber field in the BankAccount entity

**4. Make the necessary changes to the AccountNumber Text Cell**

- a) Make the necessary changes to the AccountNumber Text Cell in BankAccountsLV.pcf to display the correct watermark.

**5. Deploy your changes**

- a) In Guidewire Studio, restart (stop then debug) the server



**Hint**

For an overview of the syntax for validator patterns, see the Hint located in the *Configure the pattern* section previously in this module on Validation. For a complete listing of syntax for validator patterns, consult the Configuration Guide for your product.

### 6.3.2 Verification



**Activity**

Verify the work you have done.

**1. Verify the entity field validator behavior**

- a) In TrainingApp, add a new bank account for the Burlingame Saab contact
- b) Verify the ##### watermark displays for the Account Number Text Cell
- c) For the account number, try to enter A1-B234
- d) Verify that validator error shows with the message you created
- e) Enter 12-3456 as the account number
- f) Click Update
- g) Confirm that a new bank account was added for Burlingame Saab without an error message

## 6.4 Lab: Validation rules

*"We have to make sure that the 'Married filing jointly' and 'Married filing separately' tax filing statuses are only allowed when the Marital Status of the person is Married. We want the following behavior to be enforced: if the Tax Filing Status is set to Married filing jointly/Married filing separately and the Marital Status is NOT married then the save must be rejected with an error message and the Tax Filing Status field must be highlighted. The error message must be the following: 'Married filing jointly/Married filing separately is only allowed when contact is married.'. The error message must be localizable." – Insurance company business analysts*

### 6.4.1 Configuration

#### 1. Create a new Validation Rule

- a) To implement this user story, use the existing ABContactValidationRules Rule Set
- b) Reject the save using the `loadsave` validation level

#### 2. Deploy your changes

- a) To deploy the new rule, select Run menu → Reload changed classes in Studio
- b) To deploy the new display key, use ALT + SHIFT + L in the browser



### Review

Working with Gosu Rules

If needed, review how to work with Gosu Rules, Rule Set Categories, and Rule Sets. Refer to the Gosu Rules lesson in the InsuranceSuite Fundamentals – Kickstart course.

## 6.4.2 Verification



### Activity

Verify the work you have done

#### 1. Verify the entity field validator behavior

- a) In TrainingApp, search for Eric Andy
- b) Go to the Details page
- c) Click Edit
- d) Set the Tax Filing Status to Married filing jointly
- e) Set the Marital Status to Single
- f) Click Update
- g) Verify that you see the error message and the field is highlighted
- h) Set the Marital Status to Married
- i) Click Update
- j) Verify that the data was successfully saved



## 6.5 Lab Solution: Field-level validation in the UI



### Solution

Exact details on how to complete the lab.

#### 1. Configure the inputMask property of the widget.

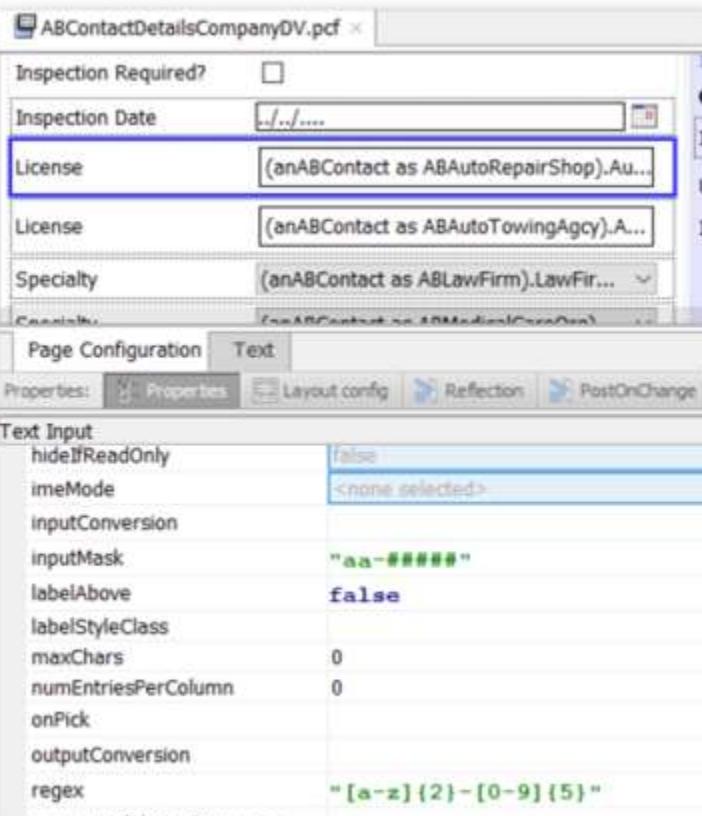
- Type “a#-#####” in the inputMask field.

The screenshot shows a PDF form titled "ABContactDetailsCompanyDV.pdf". The form contains several fields: "Inspection Required?", "Inspection Date", "License", "Specialty", and "Comments". The "License" field is currently selected and highlighted with a blue border. In the bottom right corner of the form, there is a "Page Configuration" tab and a "Text" tab. Below these tabs, there is a "Properties" panel with several properties listed:

| Property        | Value           |
|-----------------|-----------------|
| hideIfReadOnly  | false           |
| imeMode         | Keypad selected |
| inputConversion |                 |
| inputMask       | "a#-#####"      |
| labelAbove      | false           |

## 2. Configure the pattern

- Configure the regex property of the widget.



```
"[a-z]{2}-[0-9]{5}"
```

### 3. Configure the validationExpression property of the widget.

The screenshot shows the InsuranceSuite configuration interface. A Text Input widget is selected. In the Properties tab, the validationExpression property is configured with the following value:

```
(anABContact as ABAutoRepairShop).AutoRepairLicense.contains("777") ? DisplayKey.get("Ext.License777Invalid") :null
```

Below the configuration interface, a code editor window displays the same validation expression:

```
(anABContact as ABAutoRepairShop).AutoRepairLicense.contains("777") ?  
DisplayKey.get("Ext.License777Invalid") :null
```

## 6.6 Lab Solution: Field-level validation in the Data Model



### Solution

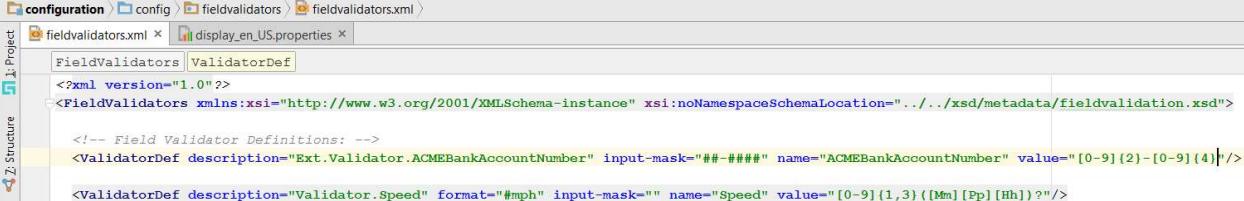
Exact details on how to complete the lab.

#### 1. Create the display key

The screenshot shows the Guidewire Studio configuration interface. A properties file named "display\_en\_US.properties" is open. The file contains the following entries:

```
Ext.Validator.ACMEPolicyNumber={0} must be a valid policy number. Example: AA-1234567
Ext.Validator.ACMEBankAccountNumber={0} A bank account number must be two numbers, a hyphen, and then four numbers. For example, 12-3456.
Ext.WebAddress=Web Address
```

## 2. Create a field validator

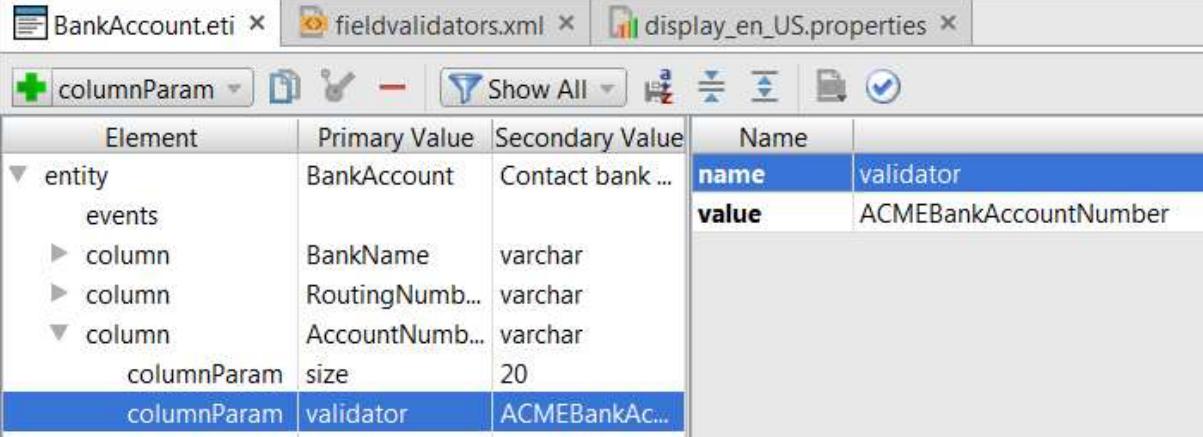


```

<?xml version="1.0"?
<FieldValidators xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../xsd/metadata/fieldvalidation.xsd">
    <!-- Field Validator Definitions: -->
    <ValidatorDef description="Ext.Validator.ACMEBankAccountNumber" input-mask="#-####" name="ACMEBankAccountNumber" value="[0-9]{2}-[0-9]{4}"/>
    <ValidatorDef description="Validator.Speed" format="#mph" input-mask="" name="Speed" value="[0-9]{1,3} ([Mm] ([Pp] [Hh]))?"/>

```

## 3. Associate the field validator with the AccountNumber field of the BankAccount entity



| Element     | Primary Value  | Secondary Value  | Name         |                       |
|-------------|----------------|------------------|--------------|-----------------------|
| entity      | BankAccount    | Contact bank ... | <b>name</b>  | validator             |
| events      |                |                  | <b>value</b> | ACMEBankAccountNumber |
| column      | BankName       | varchar          |              |                       |
| column      | RoutingNumb... | varchar          |              |                       |
| column      | AccountNumb... | varchar          |              |                       |
| columnParam | size           | 20               |              |                       |
| columnParam | validator      | ACMEBankAc...    |              |                       |

## 6.7 Lab Solution: Validation rules



### Solution

Exact details on how to complete the lab.

## 1. Create a new Rule in ABContactValidationRules.



```

ABContactValidationRules.grs x
└─ ABContact Validation Rules
    └─ Default Validation Rules
        └─ Married Tax Filing Status

USES:
uses gw.api.locale.DisplayKey

CONDITION (aBContact : entity.ABContact):
return aBContact typeis ABPerson
and
(aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_JOINT
or aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_SEPARATE
)
and
aBContact.MaritalStatus != MaritalStatus.TC_MARRIED

ACTION (aBContact : entity.ABContact, actions : gw.rules.Action):
aBContact.rejectField("TaxFilingStatus",
    ValidationLevel.TC_LOADSAVE, DisplayKey.get("Ext.Validation.MarriedTaxFilingStatus"),
    null, null)

END

```

- a) You also have the option to copy and paste it from below. However, it is recommended that you type the code based on the screenshot. **Important:** please make sure that you copy and paste the sections individually without the parts highlighted in a darker gray.

```

USES:
uses gw.api.locale.DisplayKey

CONDITION (aBContact : entity.ABContact):
return aBContact typeis ABPerson
and
(aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_JOINT
or aBContact.TaxFilingStatus == TaxFilingStatusType.TC_MARRIED_SEPARATE
)
and
aBContact.MaritalStatus != MaritalStatus.TC_MARRIED

ACTION (aBContact : entity.ABContact, actions : gw.rules.Action):
aBContact.rejectField("TaxFilingStatus",
    ValidationLevel.TC_LOADSAVE,
DisplayKey.get("Ext.Validation.MarriedTaxFilingStatus"),
    null, null)

END

```

**2. Add the display key**

- a) Open the display.properties file
- b) Add a new entry as listed below:

```
Ext.Validation.MarriedTaxFilingStatus= Married filing jointly/Married filing separately is only  
allowed when contact is married.
```

## Lesson 7 Input Sets

*"In the data model, the ABContact entity has two fields to store email addresses. However, currently, the UI only displays one email address for companies and none for persons. We want to be able to see and edit both the primary and secondary email addresses. Both email addresses should be displayed on the Summary page and Phone & Addresses card of the contacts." – Insurance company business analysts*

In this lab, you will create an Input Set. Next, you will add widgets for a contact's main and alternate email addresses to the Input Set. Then, you will add your Input Set to various PCF files.

The diagram illustrates the creation of an Input Set for managing email addresses. It shows three main components:

- Summary Page:** Displays basic contact information like Name, Public ID, and Primary Address details. The 'Email Address' field is highlighted.
- Details Page:** Displays more detailed contact information like Phone Numbers and Primary Address. The ZIP Code field is highlighted.
- Email Addresses Input Set:** A separate component containing 'Main' and 'Alternate' email address fields, which are connected via red arrows to the corresponding fields on the Summary and Details pages.

## 7.1 Prerequisites

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser. <http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 7.2 Lab: Create an Input Set

As a configuration developer, you want to be able to create Input Sets so that you can reuse a set of widgets together and/or control the visibility/editability of the set widgets. In this part of the lab, you will create the reusable EmailAddressInputSet.pcf.

**1. Create and configure the EmailAddressInputSet PCF file:**

- a) Create the Input Set in the `traininglabs` folder
- b) Specify a root object of the type ABContact
- c) Add the following widgets:
  - Add a label widget that displays “Email Addresses”
  - Add an editable input widget labeled “Main” that displays ABContact’s EmailAddress1 field
  - Add an editable input widget labeled “Alternate” that displays ABContact’s EmailAddress2 field

**2. Use Studio’s code generation feature to process the PCF file and generate all the resources**

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
  - Verify that you can see generated resources (.gs, .pcf, Expressions.gs)



### Hint

Configuring input widgets

Remember, you will have to select a specific input widget based on the data type of the field. You can always use the Data Dictionary, the generated Java class, and the widget reference table to find out the name, database type and Gosu type of the field that needs to be displayed.



### Hint

Required variables vs variables

The required variables tab defines the input parameters for a container, while the variables tab defines local variables to store values temporarily. (E.g. storing the result of an expensive function call to use it at multiple places in the PCF)

## 7.3 Lab: Reference the Input Set

In this lab, you will add the EmailAddressInputSet to various PCF files.

**1. Add the EmailAddressInputSet to ABContactSummaryDV**

- a) Navigate to ABContactSummaryDV
- b) Add an Input Divider below the Primary Address section in the same Input Column
- c) Reference the EmailAddressInputSet directly below the Input Divider

**2. Add the EmailAddressInputSet to ABPersonDetailsPage**

- a) Navigate to ABPersonDetailsPage and select the Phone & Addresses card
- b) Reference the EmailAddressInputSet directly below the Primary Address section
- c) Note: Add additional parent containers as needed.

**3. Add the EmailAddressInputSet to ABCompanyDetailsPage**

- a) Navigate to ABCompanyDetailsPage and select the Phone & Addresses card
- b) Reference the EmailAddressInputSet directly below the Primary Address section
- c) Note: add additional parent containers as needed.



### Hint

Add additional parent containers as needed

An Input Set or Input Set Ref cannot be directly added to a secondary container (e.g. Card View). The only container that can contain an Input Set is the Detail View.

### 7.3.1 Verification



#### Activity

Verify the work you have done  
Log in to TrainingApp as Alice Applegate

- 1. Use the appropriate shortcut to deploy the changes**
- 2. Edit the William Andy contact**
  - a) Navigate to William Andy
  - b) In the Summary, edit the contact
  - c) Enter the Main email for William Andy, e.g., [wiliam@guidewire.com](mailto:wiliam@guidewire.com)
  - d) Enter an Alternate email for William Andy, e.g., [wiliam-andy@gudewire.com](mailto:wiliam-andy@gudewire.com)
  - e) Click update
- 3. View the William Andy contact details**
  - a) In the sidebar menu, click Details
  - b) In Details, click Phone & Addresses
  - c) Verify that you see the email addresses that you entered

The screenshot shows a user interface for a 'Person' record named 'William Andy'. The top navigation bar includes 'TrainingApp', 'Search | ▾', and 'Contact | ▾'. A green 'Actions' button is on the left. The main area has a 'Summary' tab selected, with sub-tabs 'Basics' (highlighted in blue), 'Social Media', and 'Analysis'. A button labeled 'Suggest Least Busy User' is visible. The 'Basics' section contains the following information:

|               |                         |
|---------------|-------------------------|
| Name          | William Andy            |
| Public ID     | ab:5                    |
| Assigned User | Alice Applegate         |
| Email Address | willandy@albertsons.com |
| Created On    | 06/19/2018              |

The 'Primary Address' section includes:

|              |                                     |
|--------------|-------------------------------------|
| Address      | 345 Fir Lane<br>La Canada, CA 91352 |
| Address Type | Home                                |
| Description  |                                     |
| Valid Until  |                                     |

The 'Email Addresses' section includes:

|           |                         |
|-----------|-------------------------|
| Main      | willandy@albertsons.com |
| Alternate | wandy245@gmail.com      |

TrainingApp Search | Contact |

Actions Person: William Andy

Summary Details

Details Person Info Phone & Addresses Bank Accounts

Addresses (3)

Notes (5)

Social Media

Analysis

Interactions

History

Phone Numbers

|               |            |
|---------------|------------|
| Primary Phone | Mobile     |
| Fax Phone     |            |
| Home Phone    |            |
| Work Phone    | 2096662483 |

Primary Address

|              |                                     |
|--------------|-------------------------------------|
| Address      | 345 Fir Lane<br>La Canada, CA 91352 |
| Address Type | Home                                |
| Description  |                                     |
| Valid Until  |                                     |

Email Addresses

|           |                         |
|-----------|-------------------------|
| Main      | willandy@albertsons.com |
| Alternate | wandy245@gmail.com      |



## 7.4 Lab Solution: Create an Input Set

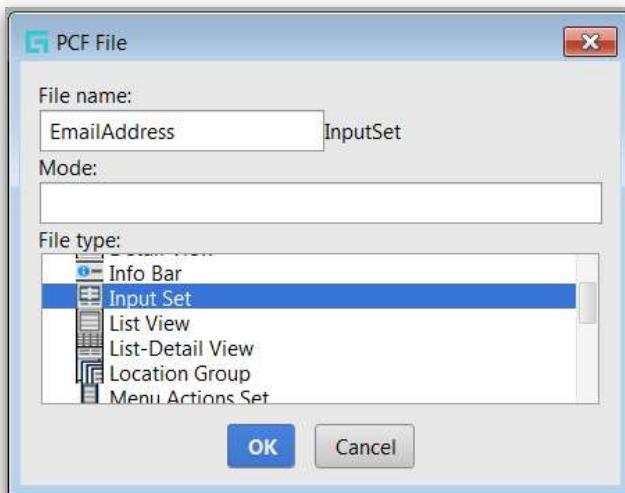


### Solution

Exact details on how to complete the lab.

1. **Create traininglabs PCF folder if it doesn't exist**

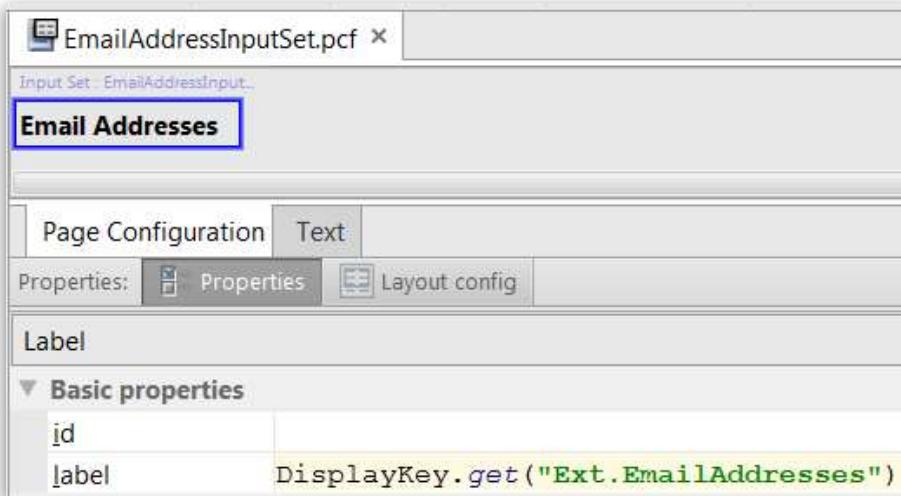
## 2. Create the Input Set



## 3. Specify the root object

A screenshot of a software interface for editing PCF files. The title bar says "EmailAddressInputSet.pcf". The main area shows a single row with the label "Input Set: EmailAddressInput...". Below this is a "Page Configuration" tab and a "Text" tab. A toolbar with buttons for "Properties", "Variables", "Referenced Widgets", "Code", and "Required Variables" is visible. The "Required Variables" tab is active. A table titled "Require" lists two entries: "anABContact" with "name\*" set to "anABContact" and "type\*" set to "ABContact".

#### 4. Add and configure the label



#### 5. Add and configure the Text Input widgets

The screenshot shows the Guidewire Studio interface for an input set named "EmailAddressInputSet.pcf". It displays two "Text Input" components under the "Email Addresses" section. Both components have their "Properties" tabs selected.

- Main Text Input:** The "label" property is set to `DisplayKey.get("Ext.Main")`. The "value\*" property is set to `anABContact.EmailAddress1`.
- Alternate Text Input:** The "label" property is set to `DisplayKey.get("Ext.Alternate")`. The "value\*" property is set to `anABContact.EmailAddress2`.

## 7.5 Lab Solution: Reference the Input Set



### Solution

Exact details on how to complete the lab.

1. Open the ABContactSummaryDV
  - a) Add an Input Divider
  - b) Add and configure the Input Set Ref widget

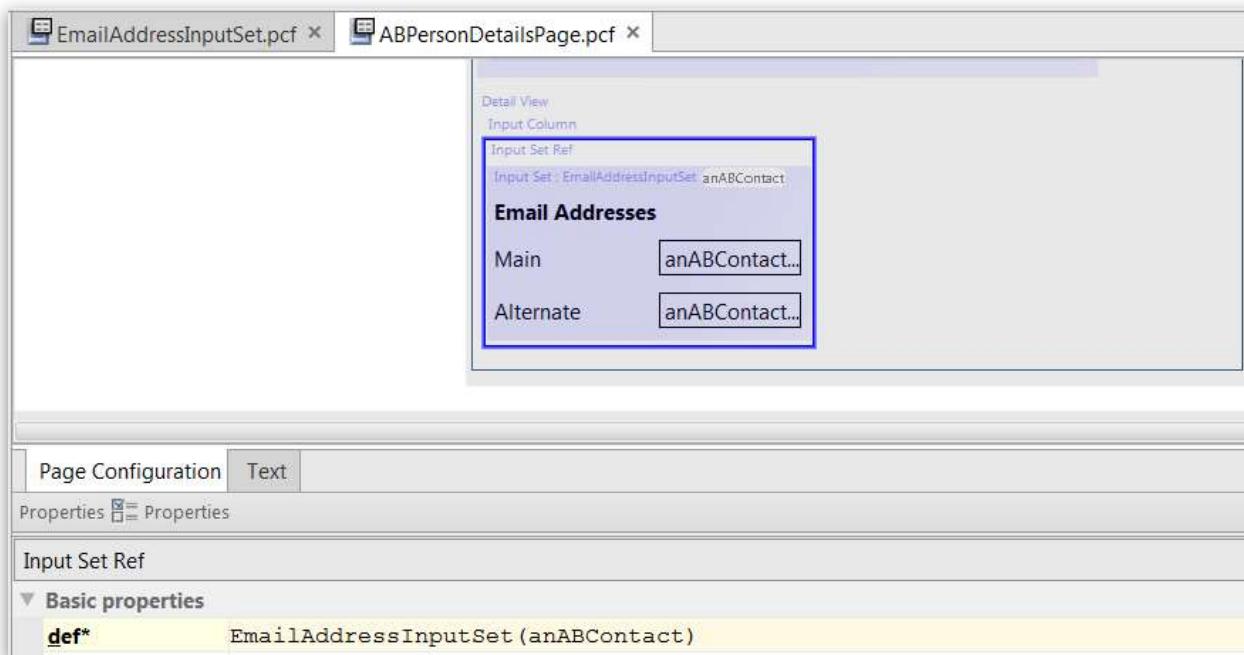
The screenshot shows the configuration interface for the ABContactSummaryDV page. At the top, there are two tabs: 'EmailAddressInputSet.pcf' and 'ABContactSummaryDV.pcf'. The main content area has several sections:

- Description:** AddressWhen/Address/Description
- Valid Until:** ../../....
- Email Addresses:** This section contains two rows:

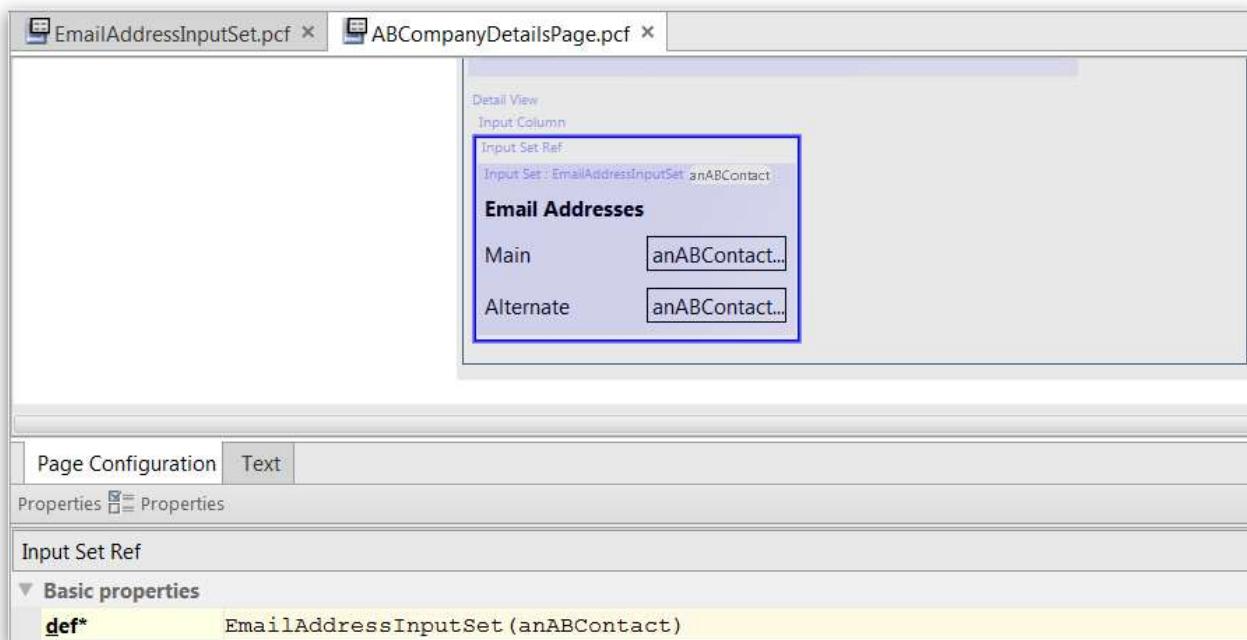
|           |                |
|-----------|----------------|
| Main      | anABContact... |
| Alternate | anABContact... |
- Page Configuration:** Text
- Properties:** Properties
- Input Set Ref:** This section is expanded, showing:
  - Basic properties:** def\* EmailAddressInputSet (anABContact)

2. Navigate to the Phone & Addresses card of the ABPersonDetailsPage. Then add and configure the InputSetRef widget.

**Note:** Ensure you add an InputSetRef from the InputLayout category and not an Input Set Ref from the Detail View Tile category. You can check you've used the correct PCF element by highlighting the element and looking in the text tab to ensure it shows as `<InputSetRef/>` and not `<TileInputSetRef/>`.



3. Navigate to the Phone & Addresses card of the ABCompanyDetailsPage. Then add and configure the InputSetRef widget.



# Lesson 8 Partial Page Update

As a configuration developer, you want to be able to configure the Post On Change behavior of widgets, so that you can improve data entry efficiency and reduce page refresh.

In this lab, you will make several basic and advanced configuration changes in order to improve both application performance and usability by configuring targeted Post On Change for various widgets in TrainingApp.

## 8.1 Prerequisites

You must first complete the following previous lesson(s):

- Creating New Entities
- List Views
- Editable List Views
- TypeLists
- Popups

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser.

<http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 8.2 Lab: Targeted Post On Change – DATA\_ONLY

*"On the Person Info card, we want to see the employer's work phone number below the employer field. We want the employer's work phone widget to be read-only and always visible. The users must see the correct employer's work phone number as soon as the value of the employer field is changed. When configuring the dynamic widget behavior, keep in mind to transfer only data to achieve the best UI performance." – Insurance company business analysts*

The screenshot shows the 'Details' screen for a person named William Andy. The 'Employer's Work Phone' field is highlighted with a yellow box and a red arrow pointing to it. The value '4084443639' is displayed in the field. The 'Employer' field is set to 'Albertson's'. Other fields shown include Full Name (William Andy), Prefix (Mr.), First Name (William), Middle Name ( ), Last Name (Andy), Suffix (Jr.), Tax ID (\*\*\*\*\*), Occupation (Sales manager), and Driver's License ( ).

In this lab, you will add a read-only input widget to the ABContactDetailsPersonDV. First, you will configure the widget to display the employer's work phone number for a given person. Next, you will make the necessary configurations so that when the value of the Employer widget changes, the read-only input widget immediately shows the new employer work phone number.

### 8.2.1 Configuration

#### 1. Add a new input widget

- In the ABContactDetailsPersonDV, in the Employment Info section, add a read-only input widget labeled "Employer's Work Phone" that displays the employer's work phone number for a given person

## 2. Configure targeted Post On Change

- a) Configure the Employer widget's targeted Post On Change properties so that only data is refreshed for the page



### 8.2.2 Verification

#### Activity

Verify the work you have done

1. Log in to TrainingApp as Alice Applegate
2. Use the appropriate shortcut to deploy the changes
3. Navigate to William Andy's Details screen
4. Click Edit and change the selection in the Employer field
5. Verify that Employer's Work Phone field changes
6. Click Cancel

## 8.3 Lab: Advanced targeted Post On Change – layout re-render

*"In the interaction popup, the Initiated By Contact, Channel Type, Summary and User fields should be visible only if the Date is set. If the date is not set or the user deletes the date then only the Date and Reason fields should be visible. The four fields should become visible/hidden as soon as the user makes a change to the Date field. Keep in mind performance when implementing this change." – Insurance company business analysts*

The screenshot shows the Guidewire TrainingApp interface. The top navigation bar includes the logo, 'TrainingApp', 'Search', and 'Contact'. The left sidebar menu has items: 'Summary', 'Details', 'Addresses (3)', 'Notes (5)', 'Social Media', 'Analysis', 'Interactions' (which is selected and highlighted in blue), and 'History'. The main content area is titled 'Interaction' with a 'Return to Interactions' button. It displays the following fields:

- Date: 09/24/2018
- InitiatedByContact: Yes (radio button selected)
- Channel Type: Email
- Interaction Summary: Quarterly courtesy contact
- Reason: <none>

A note above the interaction form says 'Person: William Andy'.

In this lab, you will modify the InteractionPopup. First, you will add an input set as a container for several of the existing widgets. Next, you will write an expression for an input set property that shows the widget when there is an InteractionDate value. Then, you will configure targeted Post On Change to re-render the layout and show the widgets in the input set when there is an InteractionDate value.

### 8.3.1 Investigation

#### 1. View the current widget behavior

- Navigate to William Andy
- In the sidebar menu, select Interactions
- Click Edit and then click Add
- Complete the cells for an interaction

- e) Click Update
- f) Click the Summary of the new interaction to open the popup
- g) Complete the blank fields of the interaction

## 2. Open the PCF to configure in Guidewire Studio

- a) To open a PCF, use ALT + SHIFT + E

### 8.3.2 Configuration

#### 1. Add a new Input Set widget

- a) In the InteractionPopup, add an Input Set widget below the Interaction Date input widget
- b) Specify the `id` property for the Input Set widget, e.g., `InteractionInputSetID`

#### 2. Move widgets into the Input Set

- a) Move the following widgets into the Input Set:
  - Initiated By Contact
  - Channel Type
  - Summary
  - User

#### 3. Modify an Input Set widget property

- a) Write an expression for the `visible` property to show the Input Set only when there is a value for the Interaction Date field

#### 4. Configure targeted Post On Change

- a) Configure the required targeted Post On Change properties for a widget so that when a date is entered, the Input Set widget immediately becomes visible

### 8.3.3 Verification



#### Activity

Add a title and replace icon if needed.

Log in to TrainingApp as Alice Applegate

1. Use the appropriate shortcut to deploy the changes
2. Create an Interaction for William Andy
3. Navigate to William Andy's Details screen
4. In the sidebar menu, select Interactions
5. Click Edit and then click Add
6. Complete the cells for an interaction
7. Click Update
8. Click the Summary of the new interaction to open the popup and verify that all fields are visible
9. Complete the blank fields of the interaction
10. Delete the Interaction Date and click outside of the date field
11. Verify that the once visible fields are now hidden



## 8.4 Lab Solution: Configure targeted Post On Change – DATA\_ONLY



### Solution

Exact details on how to complete the lab.

1. In ABContactDetailsPersonDV.pcf, add a new input widget to display the Employer's Work Phone

The screenshot shows the configuration of a detail view for ABContactDetailsPersonDV. The 'Input Column' section is expanded, showing columns for Name (Full Name, Prefix, First Name, Middle Name) and Employment Info (Occupation, Employer, Employer's Work Phone). The 'Employer's Work Phone' column is selected and highlighted with a blue border. The 'Text' tab is selected under 'Page Configuration'. In the 'Properties' panel, the 'Text' tab is active. Under 'Basic properties', the 'value\*' field is set to `(anABContact as ABPerson).Employer.WorkPhone`.

| Name        | Employment Info   |
|-------------|---|
| Full Name   | <code>(anABContact as ABPerson).FullNa...</code>              |
| Prefix      | <code>(anABContact as ABPerson).Pre...</code>                 |
| First Name  | <code>(anABContact as ABPerson).FirstNa...</code>             |
| Middle Name | <code>(anABContact as ABPerson).Middle...</code>              |
|             | <code>(anABContact as ABPerson).Occup...</code>               |
|             | <code>(anABContact as ABPerson).Em...</code>                  |
|             | <code>(anABContact as ABPerson).Employ...</code>              |
|             | <code>(anABContact as ABPerson).Employer's Work Pho...</code> |

Properties:

|            |               |            |              |
|------------|---------------|------------|--------------|
| Properties | Layout config | Reflection | PostOnChange |
|------------|---------------|------------|--------------|

Text Input

Basic properties

|               |   |
|---------------|---|
| editable      | false   |
| <b>id*</b>    | EmployersWorkPhone  |
| label         | DisplayKey.get("Ext.EmployersWorkPhone")                  |
| required      |   |
| <b>value*</b> | <code>(anABContact as ABPerson).Employer.WorkPhone</code> |

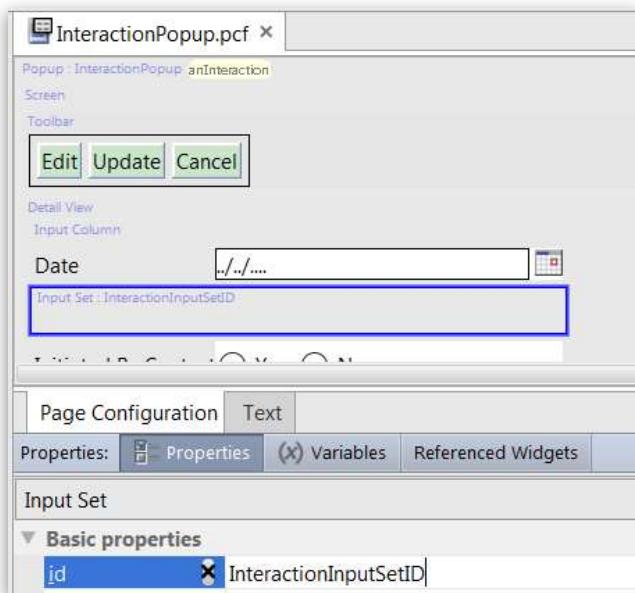
## 8.5 Lab Solution: Advanced targeted Post On Change – layout re-render



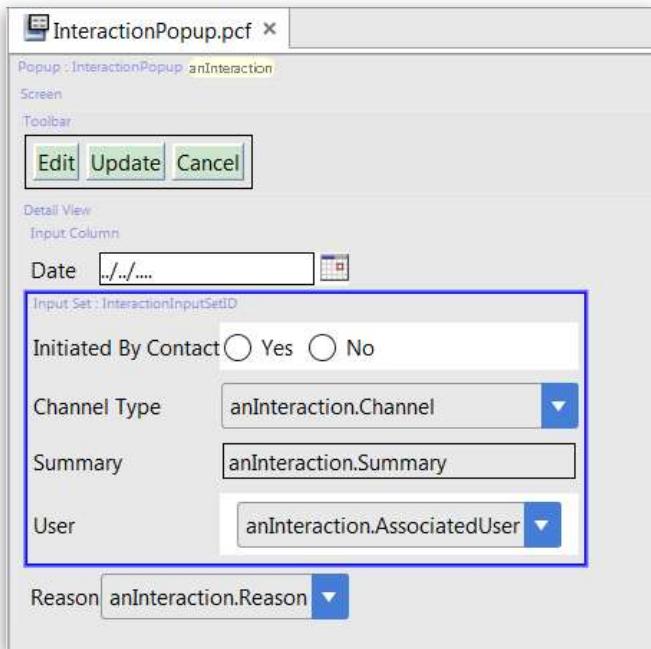
### Solution

Exact details on how to complete the lab.

1. Add a new Input Set widget into InteractionPopup under the Interaction Date field and then specify the id property



## 2. Move input widgets into the Input Set

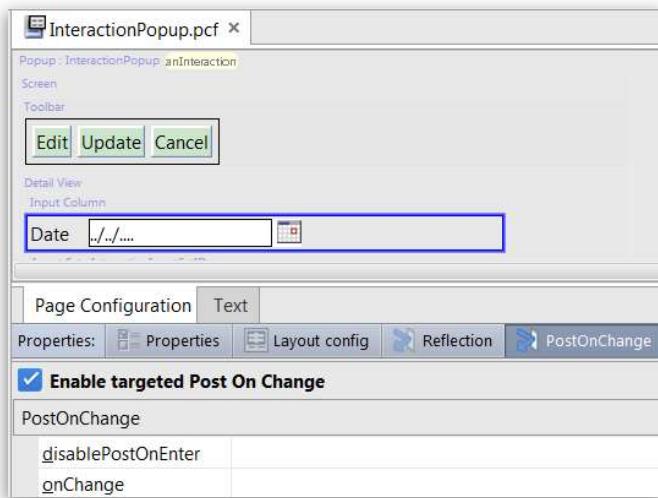


3. Write an expression for the **visible** property to show the Input Set only when there is a value for the Interaction Date field

The screenshot shows the Guidewire Studio interface for creating a page configuration file (PCF). The main window displays a form with various input fields and dropdown menus. Below the form is a properties panel with tabs for 'Page Configuration' and 'Text'. The 'Properties' tab is active, showing sections for 'Basic properties' and 'Advanced properties'. In the 'Advanced properties' section, the 'visible' property is set to the expression `anInteraction.InteractionDate != null`.

| Properties     |  |
|----------------|--|
| Property       | Value  |
| id             | InteractionInputSetID                              |
| mode           |  |
| desc           |  |
| editable       |  |
| hideIfEditable | false  |
| hideIfReadOnly | false  |
| visible        | <code>anInteraction.InteractionDate != null</code> |

4. Configure targeted Post On Change so that when a date is entered, the Input Set widget immediately becomes visible



# Lesson 9 Subtypes

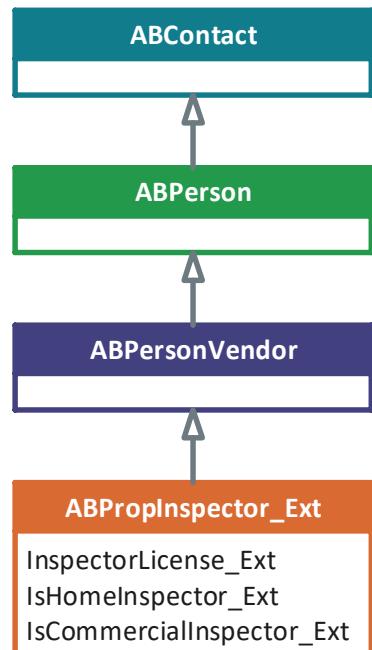
In this lab, you will use the Entity Editor in Guidewire Studio to modify the TrainingApp data model. You will create a new custom subtype entity. Then, you will extend an existing base application subtype entity.

## 9.1 Prerequisites

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser. <http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 9.2 Lab: Create a new custom subtype entity

*"We need to store information about Property Inspectors. After reviewing the data model, we found that there is currently no entity available to do this. Configure the data model and create the new a subtype entity as specified in the diagram and table below."* – Insurance company business analysts



In this lab, you will create a subtype entity for Property Inspectors using ABPersonVendor as the supertype entity. For each new entity element, remember to set the nullok attribute to true if not specifically defined to be false. When required, add an element description and specify column parameters.

## 1. Create the **ABPropInspector\_Ext** subtype entity

- Create a new ETI file. Specify the following values in the new entity dialog:
  - Entity: **ABPropInspector\_Ext**
  - Entity Type: **subtype**
  - Supertype: **ABPersonVendor**

## 2. In the Entity Editor, add the following fields:

| Field Name                | Datatype                        | Null ok? |
|---------------------------|---------------------------------|----------|
| InspectorLicense_Ext      | a string of up to 40 characters | true     |
| IsHomeInspector_Ext       | a boolean value                 | true     |
| IsCommercialInspector_Ext | a boolean value                 | true     |

## 1. Use Studio's code generation feature to process the entity and generate the Java class

- If possible, use incremental code generation
- Verify that there were no errors during code generation

Open the generated Java class and verify that the class extends the ABPersonVendor and you can see the three new properties



### Best Practices

Use \_Ext in subtypes name and field names

Guidewire recommends the use of the \_Ext suffix (or any unique suffix) in the entity name when creating new (subtype) entities.

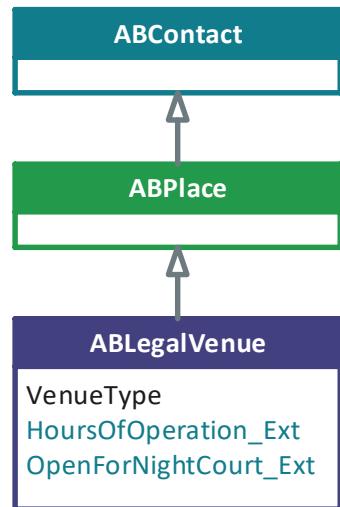
As for fields, it depends on whether the new subtype entity is part of a base application subtype hierarchy or a custom subtype hierarchy.

- The \_Ext suffix is recommended if the subtype is part of a base application subtype hierarchy

- The \_Ext suffix is not needed if the subtype is part of a custom subtype hierarchy
- In this lab, ABPersonVendor is the Supertype of ABPropInspector\_Ext. ABPersonVendor is a base application subtype; thus it is part of a base application subtype hierarchy. Guidewire can add new fields to base application subtypes in future releases. These new fields would be inherited and could cause conflict in ABPropInspector\_Ext if the \_Ext (or any unique suffix) is not used in the field names.

## 9.3 Lab: Extend an existing base application subtype entity

*We are going to use the ABLegalVenue OOTB subtype entity to store information about legal venues. However, during our initial analysis, we identified a gap and found that we cannot store the following two pieces of information: (1) the hours of operation and (2) if the legal venue is open for night court or not. Configure the data model and create the new a subtype entity as specified in the diagram and table below.” - Insurance company business analysts*



In this lab, you will create an entity extension of ABLegalVenue, an existing subtype entity. You will create new entity elements to capture specific information for legal venues. For each new entity element, remember to set the nullok attribute to true if not specifically defined to be false. When required, add an element description and specify column parameters.

### 2. Create the ABLegalVenue entity extension

- a) Create ABLegalVenue.etx

**3. In the Entity Editor, add the following fields:**

| Field Name            | Datatype                        | Null ok? |
|-----------------------|---------------------------------|----------|
| HoursOfOperation_Ext  | a string of up to 40 characters | true     |
| OpenForNightCourt_Ext | a boolean value                 | true     |

**4. Use Studio's code generation feature to process the entity and generate the Java class**

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Open the generated Java class and verify that you can see the newly added properties



## Best Practices

Use \_Ext in entity field name when extending base application (subtype) entities

For fields that are added to a base application entity, Guidewire recommends that the field name should end with \_Ext (or start with Ext\_). This is to prevent potential conflicts during the upgrade to the next version of the Guidewire application.

### 9.3.1 Verification



## Activity

Verify the work you have done

As a configuration developer, you want to be able to properly deploy new and changed data model resources.

**1. Restart the server to deploy the changes**

- a) During server restart Studio first runs the code generators, then compiles the project and finally deploys the resources

**2. Regenerate the Data Dictionary**

**3. Open the Data Dictionary**

**4. In Windows Explorer, navigate to the data dictionary.**

5. Open the data dictionary using your preferred browser.
6. Verify your changes to the ABPropInspector\_Ext entity
7. Verify your changes to the ABLegalVenue entity



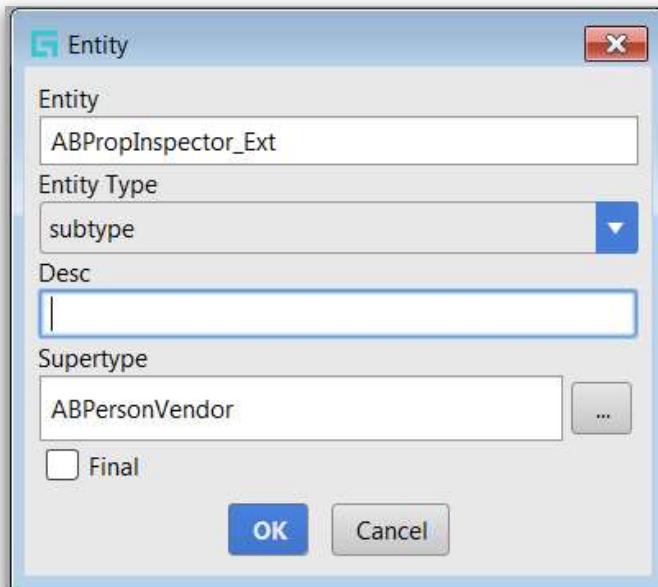
## 9.4 Lab Solution: Create a new custom subtype entity



### Solution

Exact details on how to complete the lab.

1. Create the ABPropInspector\_Ext.eti file in the ...\\configuration\\config\\Extensions\\Entity folder



## 2. Add the fields

ABPropInspector\_Ext.eti

| Element      | Primary Value        | Secondary Value | Name                             |
|--------------|----------------------|-----------------|----------------------------------|
| subtype      | ABPropInspector...   |                 | <b>name</b> InspectorLicense_Ext |
| column       | InspectorLicense_... | varchar         | <b>type</b> varchar              |
| columnP.size | 40                   |                 | <b>nullok</b> true               |

ABPropInspector\_Ext.eti

| Element | Primary Value        | Secondary Value | Name                            |
|---------|----------------------|-----------------|---------------------------------|
| subtype | ABPropInspector...   |                 | <b>name</b> IsHomeInspector_Ext |
| column  | InspectorLicense_... | varchar         | <b>type</b> bit                 |
| column  | IsHomeInspector...   | bit             | <b>nullok</b> true              |

ABPropInspector\_Ext.eti

| Element | Primary Value        | Secondary Value | Name                                  |
|---------|----------------------|-----------------|---------------------------------------|
| subtype | ABPropInspector...   |                 | <b>name</b> IsCommercialInspector_Ext |
| column  | InspectorLicense_... | varchar         | <b>type</b> bit                       |
| column  | IsHomeInspector...   | bit             | <b>nullok</b> true                    |
| column  | IsCommercialIns...   | bit             | <b>desc</b>                           |

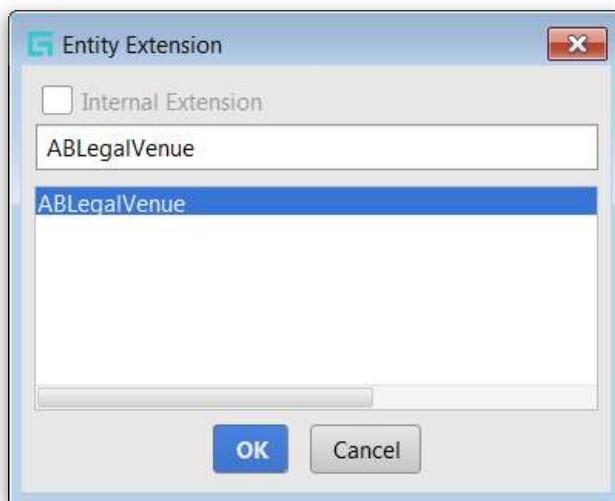
## 9.5 Lab Solution: Extend an existing base application subtype entity



### Solution

Exact details on how to complete the lab.

1. Create the ABLegalVenue.etx file in ...\\configuration\\config\\Extensions\\Entity folder

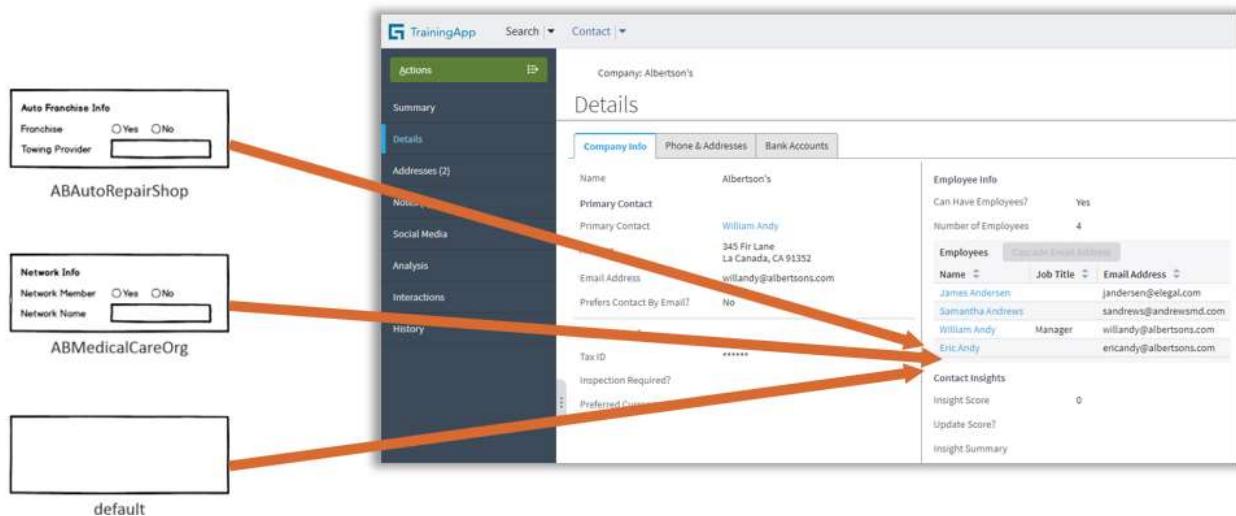


2. Add the fields

| Element   | Primary Value      | Secondary Value | Name          |                       |
|-----------|--------------------|-----------------|---------------|-----------------------|
| extension | ABLegalVenue       |                 | <b>name</b>   | OpenForNightCourt_Ext |
| column    | HoursOfOperatio... | varchar         | <b>type</b>   | bit                   |
| column    | OpenForNightCo...  | bit             | <b>nullok</b> | true                  |

## Lesson 10 Modes

*"An insurance company has additional "Specialty Info" details for ABAutoRepairShop contacts and ABMedicalCareOrg contacts that the current user interface does not display. In each case, the Company Info card should display these fields below Employees list. See the wireframe for more details." – Insurance company business analysts*



As a configuration developer, you want to be able to create and reference modal container sets, so you can design different variations for a small part of a screen. In this lab, you will create several modal Input Sets.

### 10.1 Prerequisites

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser. <http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

### 10.2 Lab: Create a set of modal Input Sets

In this part of the lab, you will create a set of modal Input Sets, each of which has an ABContact root object, in the traininglabs PCF Folder. Remember, every PCF file in the set must have the same name, same type and same required variable list.

#### 1. Create SpecialtyInfoInputSet.default.pcf :

- Create Input Set in the traininglabs folder and specify the mode "default"

- b) Specify a root object of the type ABContact
- c) This input set needs no widgets and should be empty

**2. Create the SpecialtyInfoInputSet.ABAutoRepairShop.pcf for ABAutoRepairShop contacts :**

- a) Create Input Set in the `traininglabs` folder and specify the mode “ABAutoRepairShop”
- b) Specify a root object of the type ABContact
- c) Add the following widgets:
- d) A label widget that displays “Auto Franchise Info”
- e) An editable input widget labeled “Franchise” that displays ABAutoRepairShop’s `IsFranchise` field
- f) An editable input widget labeled “Towing Provider” that displays ABAutoRepairShop’s `TowingProvider` field

**3. Create the SpecialtyInfoInputSet.ABMedicalCareOrg.pcf for ABMedicalCareOrg contacts :**

- a) Create Input Set in the `traininglabs` folder and specify the mode “ABMedicalCareOrg”
- b) Specify a root object of the type ABContact
- c) Add the following widgets:
  - A label widget that displays “Network Info”
  - An editable input widget labeled “Network Member” that displays ABMedicalCareOrg’s `IsMemberOfNetwork` field
  - An editable input widget labeled “Network Name” that displays ABMedicalCareOrg’s `NetworkName` field

**4. Use Studio’s code generation feature to process the PCF file and generate all the resources**

- a) If possible, use incremental code generation
- b) Verify that there were no errors during code generation
- c) Verify that you can see generated resources (.gs, .pcf, Expressions.gs)



### Hint

Configuring input widgets

Remember, you will have to select a specific input widget based on the data type of the field. You can always use the Data Dictionary, the generated Java class, and the widget reference table to find out the name, database type and Gosu type of the field that needs to be displayed.



### Hint

Required variables vs variables

The required variables tab defines the input parameters for a container, while the variables tab defines local variables to store values temporarily. (E.g. storing the result of an expensive function call to use it at multiple places in the PCF)

## 10.3 Lab: Reference the set of modal Input Sets

In this lab, you will reference the set of modal Input Sets.

1. **1. Add a reference to the modal container set on the Details screen's Company Info card directly under the Employees list.**
  - a) Navigate to a company contact's Details screen
  - b) Open the PCF file in Studio
  - c) Add the Input Set Ref widget below the Employees list in the correct PCF file

### 10.3.1 Verification



#### Activity

Verify the work you have done

1. Log in to TrainingApp as Alice Applegate
2. Use the appropriate shortcut to deploy the changes
3. Navigate to the Details screen of an ABAutoRepairShop (e.g. Burlingame Saab) contact and verify that the “Auto Franchise Info” is displayed
4. Navigate to the Details screen of an ABMedicalCareOrg (e.g. Health South) contact and verify that the “Network Info” is displayed
5. Navigate to the Details screen of a contact that is neither ABAutoRepairShop nor ABMedicalCareOrg (e.g. Alberson’s). Verify that the default Input Set is displayed.

Auto Repair Shop: Burlingame Saab

### Details

| Company Info         |                 | Phone & Addresses   | Bank Accounts         | Vendor Info   |
|----------------------|-----------------|---------------------|-----------------------|---------------|
| Name                 | Burlingame Saab | Employee Info       |                       |               |
| Primary Contact      |                 | Can Have Employees? | Yes                   |               |
| Primary Contact      |                 | Number of Employees | 0                     |               |
| Address              | <empty>         | Employees           | Cascade Email Address |               |
| Email Address        |                 | Name                | Job Title             | Email Address |
| Additional Info      |                 | No data to display  |                       |               |
| Tax ID               | *****           | Auto Franchise Info |                       |               |
| Inspection Required? |                 | Franchise           | Towing Provider       |               |
| License              |                 | Financial Personnel |                       |               |

Medical Care Organization: Health South

## Details

| Company Info  | Phone & Addresses | Bank Accounts | Vendor Info |
|---|-------------------|---------------|-------------|
| Name<br>Health South  |                   |               |             |
| Primary Contact   |                   |               |             |
| Primary Contact   |                   |               |             |
| Address<br><empty>  |                   |               |             |
| Email Address   |                   |               |             |
| <b>Additional Info</b>  |                   |               |             |
| Tax ID<br>*****   |                   |               |             |
| Inspection Required?  |                   |               |             |
| Specialty<br>Anesthesiology   |                   |               |             |
| <b>Employee Info</b>  |                   |               |             |
| Can Have Employees?   | Yes               |               |             |
| Number of Employees   | 0                 |               |             |
| <b>Employees</b> <input type="button" value="Cascade Email Address"/> |                   |               |             |
| Name  | Job Title         | Email Address |             |
| No data to display  |                   |               |             |
| <b>Network Info</b>   |                   |               |             |
| Network Member  |                   |               |             |
| Network Name  |                   |               |             |
| <b>Financial Personnel</b>  |                   |               |             |



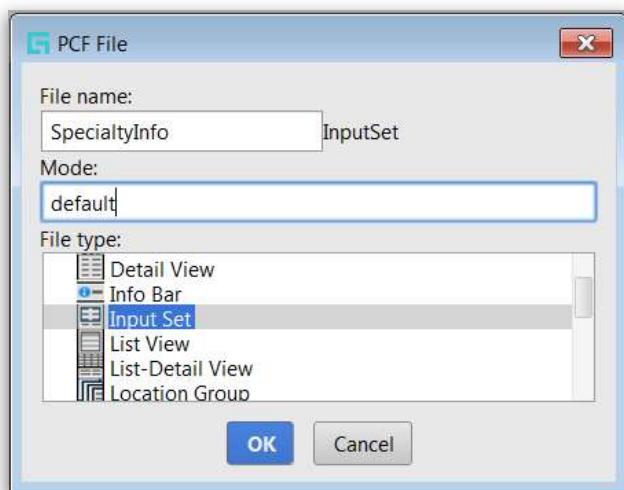
## 10.4 Lab Solution: Create a set of modal Input Sets



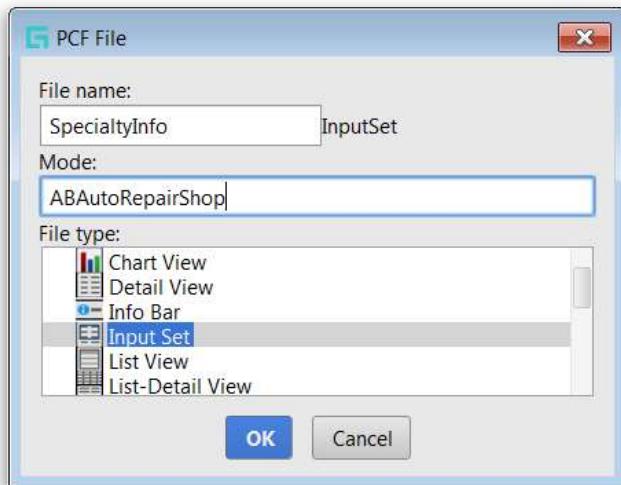
### Solution

Exact details on how to complete the lab.

1. Create traininglabs PCF folder if it doesn't exist
2. Create the default Input Set
3. Create the PCF file



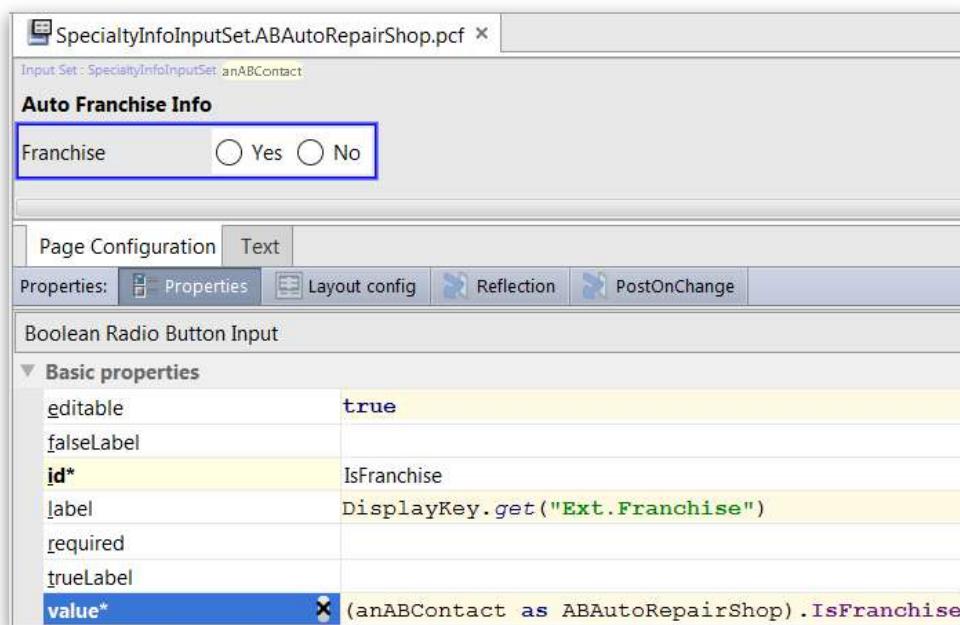
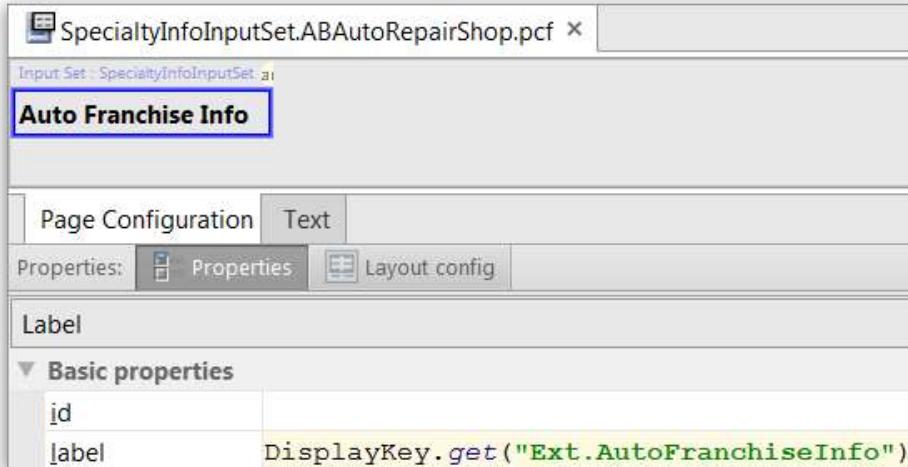
- a) Specify the root object

**4. Create the Input Set for ABAutoRepairShop contacts****a) Create the PCF file****b) Specify the root object**

The screenshot shows the PCF editor interface for the file 'SpecialtyInfoInputSet.ABAutoRepairShop.pcf'. The title bar displays the file name. The main workspace shows the root object 'Input Set : SpecialtyInfoInput...'. Below the workspace, tabs for 'Page Configuration' and 'Text' are visible, with 'Text' being the active tab. A toolbar above the tabs includes icons for Properties, Variables, Referenced Widgets, Code, and Required Variables. The 'Properties' tab is currently selected. In the bottom half of the screen, a table titled 'Require' is displayed, listing two required variables:

| anABContact | <b>name*</b> | anABContact |
|-------------|--------------|-------------|
|             | <b>type*</b> | ABContact   |

## c) Add the widgets

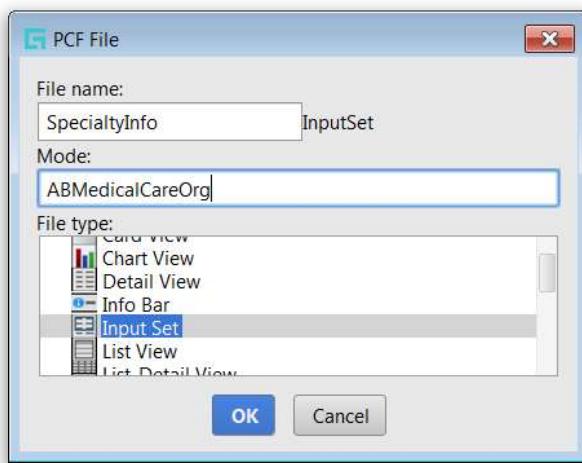


The screenshot shows the PCF editor interface for a file named 'SpecialtyInfoInputSet.ABAutoRepairShop.pcf'. The main window displays a section titled 'Auto Franchise Info' with a 'Franchise' field (radio buttons for Yes/No) and a 'Towing Provider' field (a dropdown menu labeled '(anABContact ...)'). Below this is a 'Text Input' section containing a table of properties:

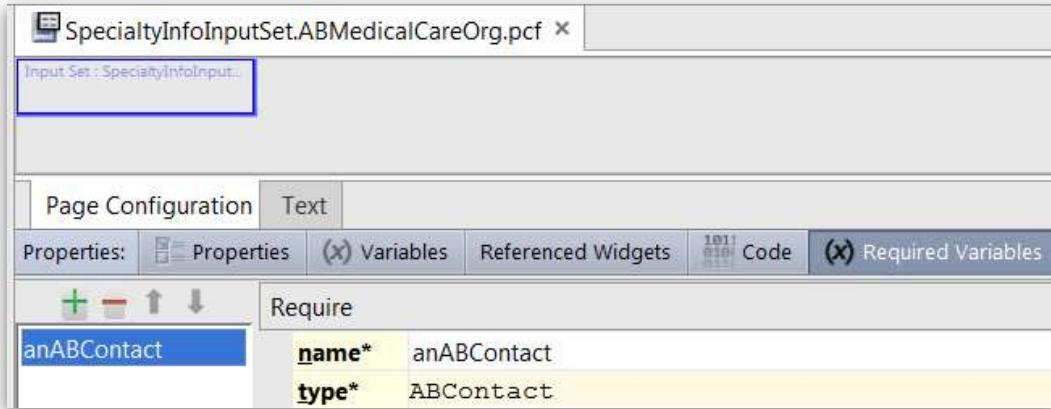
| Basic properties |  |
|------------------|--|
| editable         | true   |
| <b>id*</b>       | TowingProvider                                   |
| label            | DisplayKey.get("Ext.TowingProvider")             |
| required         |  |
| <b>value*</b>    | (anABContact as ABAutoRepairShop).TowingProvider |

## 5. Create the Input Set for ABMedicalCareOrg contacts

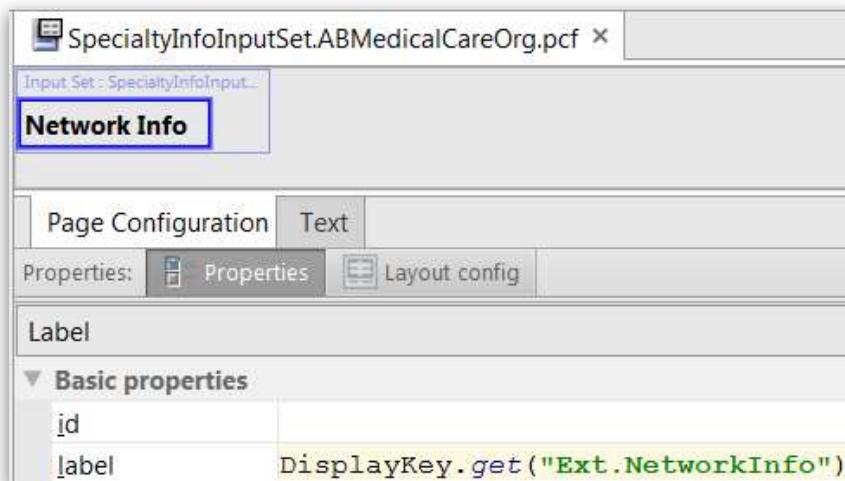
- Create the PCF file



## b) Specify the root object



## c) Add the widgets



SpecialtyInfoInputSet.ABMedicalCareOrg.pcf

Input Set : SpecialtyInfoInputSet anABCContact

**Network Info**

Network Member  Yes  No

Page Configuration Text

Properties: Properties Layout config Reflection PostOnChange

Boolean Radio Button Input

Basic properties

|               |  |
|---------------|--|
| editable      | true   |
| falseLabel    |  |
| <b>id*</b>    | NetworkMember  |
| label         | DisplayKey.get("Ext.NetworkMember")                  |
| required      |  |
| trueLabel     |  |
| <b>value*</b> | (anABCContact as ABMedicalCareOrg).IsMemberOfNetwork |

SpecialtyInfoInputSet.ABMedicalCareOrg.pcf

Input Set : SpecialtyInfoInputSet anABCContact

**Network Info**

Network Member  Yes  No

Network Name

Page Configuration Text

Properties: Properties Layout config Reflection PostOnChange

Text Input

Basic properties

|               |  |
|---------------|--|
| editable      | true   |
| <b>id*</b>    | NetworkName                                    |
| label         | DisplayKey.get("Ext.NetworkName")              |
| required      |  |
| <b>value*</b> | (anABCContact as ABMedicalCareOrg).NetworkName |

## 10.5 Lab Solution: Reference the set of modal Input Sets



### Solution

Exact details on how to complete the lab.

1. Open the ABContactDetailsCompanyDV
2. Add Input Set Ref widget below the Employees List View
3. Configure the def property

4. Specify a dynamic value for the mode property

**Employees**

**Auto Franchise Info**

| Property | Value                               |
|----------|-------------------------------------|
| def*     | SpecialtyInfoInputSet (anABContact) |
| id       |                                     |
| mode     | anABContact.Subtype                 |

# Lesson 11 Entity Names

*"We defined a new format to display interactions in the UI. This will help users to easily identify additional information about each interaction. The same format may be needed at multiple places later, so it needs to be easily reusable and performant. The Interaction date should be displayed in the string format shown in the wireframe below. Only limited information should be retrieved from the database to construct the string values for interaction date." – Insurance company business analysts*

## Required display of InteractionsLV:

As a configuration developer, you want to be able to create entity names, so that you can define reusable, UI friendly names for entity instances. In this lab, you will use the Entity Name editor to create an entity name for the Interaction entities. The interaction date field will display a string constructed in the format shown above. Then, you will modify the Interactions list view to display the entity name.

## 11.1 Prerequisites

You must first complete the following previous lesson(s):

- Creating New Entities
- List Views
- Editable List Views

- Typelists
- Popups

For this lab, use TrainingApp, Guidewire Studio, and a supported web browser. <http://localhost:8880/ab/ContactManager.do> is the default URL for TrainingApp. To view, edit, and delete various contacts, log in to TrainingApp as Alice Applegate. The login/password for Alice Applegate is aapplegate/gw.

## 11.2 Lab: Create and reference an entity name

In this lab, you will first create the entity name for the Interaction\_Ext entity. Then, you will modify the InteractionsLV.pcf file to display the entity name.

1. **Create an Interaction\_Ext entity name**
2. **Create the following variables in the variable table**
  - a) Add the `interactionDate` variable with the entity path `Interaction_Ext.InteractionDate`
  - b) Add the `channel` variable with the entity path `Interaction_Ext.Channel`
3. **Define the Default name that returns one of the following three formats using the `interactionDate` and `channel` variables**
  - a) If the date is null, then return the following:  
(No Date): ChannelType
  - b) If the date is more than one (1) day in the future, then return the following:  
(# days in the future) YYYY-MM-DD: ChannelType
  - c) Otherwise, return the following:  
YYYY-MM-DD: ChannelType
4. **Modify the InteractionsLV.pcf**
  - a) Open the InteractionsLV.pcf
  - b) Change the Date Cell widget to a Text Cell widget
  - c) Configure the Text Cell widget to...
    - be read-only
    - show the new `Interaction_Ext` entity name for the `currentInteraction` object
5. **Restart the server to deploy all the changes**

## 11.2.1 Verification



### Activity

Verify the work you have done

1. Log in to TrainingApp as Alice Applegate
2. Edit the Interactions page for William Andy
  - a) Navigate to William Andy
  - b) In the sidebar menu, click Interactions
  - c) Create an interaction in the future: enter a date that is more than one (1) day in the future; also specify Channel Type, Interaction, and Reason fields
  - d) Create an interaction with no interaction date: leave the Date field blank; also specify Channel Type, Interaction, and Reason fields

- e) Create an interaction with today's date: enter today's date for the Date field; specify Channel Type, Interaction, and Reason fields

The screenshot shows the Guidewire InsuranceSuite interface. At the top, there is a header with the logo 'TrainingApp' and search/contact dropdowns. On the left, a sidebar menu lists 'Actions', 'Summary', 'Details', 'Addresses (3)', 'Notes (5)', 'Social Media', 'Analysis', 'Interactions' (which is highlighted in blue), and 'History'. The main content area is titled 'Interactions' and shows a table of interactions for 'Person: William Andy'. The table has columns for 'Date', 'Summary', and 'Associated User'. The data in the table is as follows:

| Date                                      | Summary           | Associated User |
|---|-------------------|-----------------|
| (2 days in the future) 2018-10-13 : Email | Send reminder     | Alice Applegate |
| 2018-10-11 : Fax                          | Courtesy contact  | Bruce Baker     |
| (No Date) : Phone                         | Check Bill Status | Bruce Baker     |



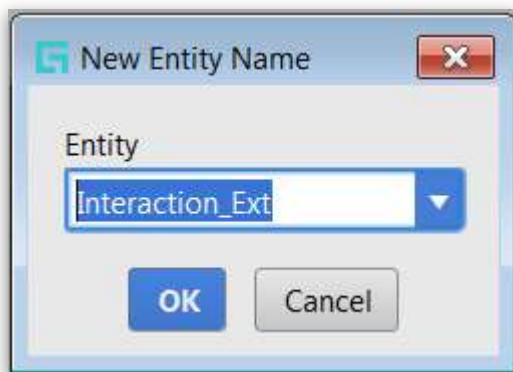
## 11.3 Lab Solution: Create and reference an Entity Name



### Solution

Exact details on how to complete the lab.

1. Create the new Interaction\_Ext entity name in the ...\\configuration\\config\\Entity Names folder



2. Create the variables in the variable table

| Name            | Entity Path                     | Sort Path | Sort Order | Use Entity Name?         |
|-----------------|---------------------------------|-----------|------------|--------------------------|
| interactionDate | Interaction_Ext.InteractionDate |           |            | <input type="checkbox"/> |
| channel         | Interaction_Ext.Channel         |           |            | <input type="checkbox"/> |

3. Create the Default name

```

uses gw.api.util.DateUtil

var name = ""

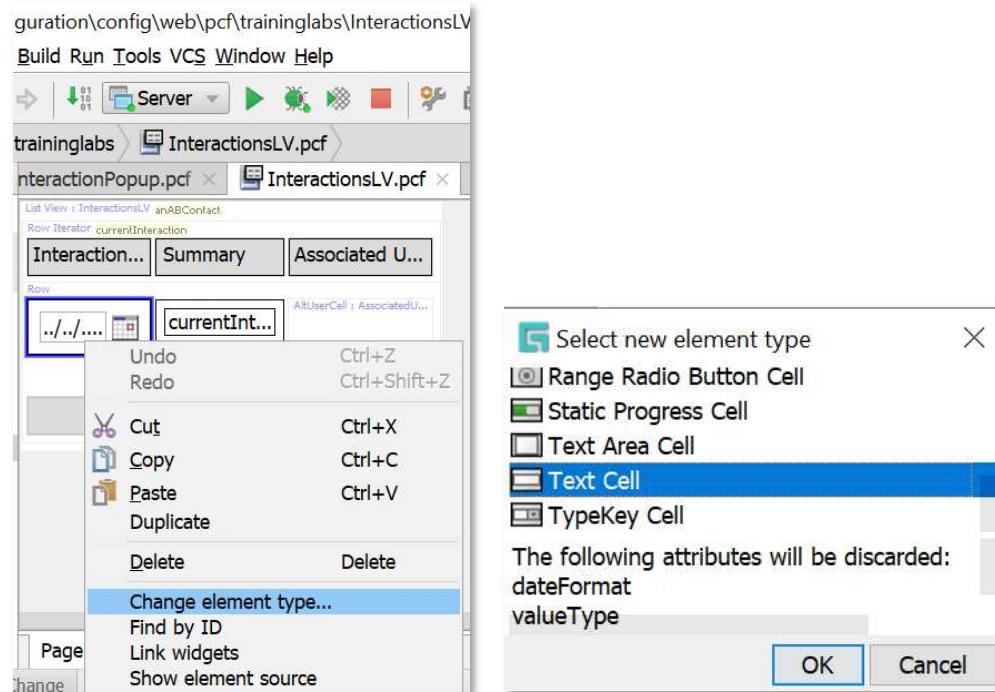
if (interactionDate == null) {
    name = "(No Date) : " + channel.DisplayName
} else {
    var numberDays = DateUtil.currentDate().differenceInDays(interactionDate)
    if (numberDays > 1) {
        name = "(" + numberDays + " days in the future) " + interactionDate + " : " +
channel.DisplayName
    } else {
        name = interactionDate + " : " + channel.DisplayName
    }
}

```

```
}
```

```
return name
```

4. **Modify the InteractionsLV.pcf – Since the Interaction Date field will now display the entity name, we'll change the InteractionDate element type from Date Cell to Text Cell.**



**5. Configure the editable and value properties.**

The screenshot shows the Guidewire Studio interface for configuring a page component. The top window title is "InteractionsLV.pcf". The page view shows a row iterator with three columns: "Date", "Summary", and "User". Below this, there are two rows, each containing two cells labeled "currentInteraction...". The bottom section is a "Text Cell" configuration panel. It has tabs for "Page Configuration" and "Text", with "Text" selected. Under "Properties", the "Properties" tab is active. A table lists the following properties:

| Property   | Value                                 |
|------------|---------------------------------------|
| action     |                                       |
| editable   | false                                 |
| <b>id*</b> | InteractionDate                       |
| label      | DisplayKey.get("Ext.InteractionDate") |
| required   |                                       |
| value      | currentInteraction.DisplayName        |

**6. Stop and restart the server to check your work.**