



**Designer Training**

**Fourth Edition (November 2006)**

This edition applies to version 2.6.1 of IBM® Workplace Forms™.

© **Copyright International Business Machines Corporation 2006. All rights reserved.**

US Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP  
Schedule Contract with IBM Corp.

# Contents

<b>1.</b>	<b>INTRODUCTION</b> .....	<b>1</b>
1.1.	PURPOSE .....	1
1.2.	AUDIENCE .....	1
<b>2.</b>	<b>BUILDING SIMPLE XFDL FORMS</b> .....	<b>3</b>
2.1.	AUDIENCE .....	3
2.2.	OVERVIEW.....	3
2.3.	WORKING WITH THE PALETTE .....	3
2.3.1.	<i>What is the Palette?</i> .....	3
2.3.2.	<i>Palette Layout</i> .....	4
2.3.3.	<i>Selection Types</i> .....	5
2.3.4.	<i>Creating Palette Drawers</i> .....	5
2.3.5.	<i>XFDL Objects</i> .....	6
2.3.6.	<i>XFDL Object Library</i> .....	7
2.4.	WORKING WITH THE CANVAS .....	9
2.5.	WORKING WITH ITEM PROPERTIES .....	10
2.5.1.	<i>Changing a property</i> .....	10
2.5.2.	<i>Resetting a property back to the default value</i> .....	13
2.5.3.	<i>Filtering the Properties View</i> .....	14
2.6.	FORM PREVIEW .....	14
2.7.	XFDL EXERCISE 1 - BUILD SIMPLE XFDL FORM .....	15
2.8.	GRID OPTIONS.....	26
2.9.	REFINING FORM LAYOUT .....	28
2.9.1.	<i>Moving Objects</i> .....	28
2.9.2.	<i>Aligning Objects</i> .....	29
2.9.3.	<i>Expanding Objects</i> .....	31
2.10.	XFDL EXERCISE 2 – ALIGNMENT .....	33
2.11.	TAB ORDER .....	36
2.12.	XFDL EXERCISE 3 - MODIFYING THE TAB ORDER .....	37
2.13.	WORKING WITH ENCLOSURES .....	43
2.13.1.	<i>Data</i> .....	43
2.13.2.	<i>JAR</i> .....	43
2.13.3.	<i>Schema</i> .....	44
2.13.4.	<i>WSDL</i> .....	44
2.13.5.	<i>XForms Instance</i> .....	44
<b>3.</b>	<b>UNDERSTANDING XFDL</b> .....	<b>45</b>
3.1.	AUDIENCE .....	45
3.2.	OVERVIEW.....	45
3.3.	NAMESPACES .....	45
3.4.	NODE STRUCTURE .....	47
3.5.	REFERENCES.....	48
3.6.	XFDL EXERCISE 4 – WRITING XFDL REFERENCES .....	49

3.7.	OPERATORS .....	50
3.7.1.	<i>Numeric</i> .....	50
3.7.2.	<i>Logical</i> .....	50
3.7.3.	<i>Miscellaneous</i> .....	50
3.7.4.	<i>Relational</i> .....	51
3.8.	COMPUTES .....	52
3.8.1.	<i>Mathematical Computes</i> .....	52
3.8.2.	<i>Conditional Computes</i> .....	52
3.8.3.	<i>Creating Computes in the Designer</i> .....	53
3.9.	XFDL EXERCISE 5 – WRITING COMPUTES .....	54
3.10.	FUNCTIONS .....	58
3.10.1.	<i>Explore XFDL Functions</i> .....	58
3.11.	XFDL EXAMPLE 6 – USING XFDL FUNCTIONS .....	59
3.12.	EVENTS .....	76
3.13.	XFDL EXERCISE 7 – USING XFDL EVENTS .....	77
3.14.	CUSTOM OPTIONS .....	91
3.14.1.	<i>Creating Re-Usable Variables</i> .....	91
3.14.2.	<i>Creating Containers for Computes</i> .....	91
3.14.3.	<i>Exercise - Creating a Custom Option in the Designer</i> .....	92
<b>4.</b>	<b>ADVANCED XFDL CONCEPTS .....</b>	<b>93</b>
4.1.	AUDIENCE .....	93
4.2.	OVERVIEW .....	93
4.3.	WORKING WITH THE SOURCE VIEW .....	93
4.3.1.	<i>What is Code Completion?</i> .....	93
4.3.2.	<i>What is folding text?</i> .....	94
4.3.3.	<i>What do the marks on the left indicate? (That appear when editing)</i> .....	94
4.3.4.	<i>What are Bookmarks?</i> .....	95
4.3.5.	<i>Validating Your Code</i> .....	95
4.3.6.	<i>Check if code is well formed</i> .....	95
4.4.	DUPLICATION .....	96
4.5.	PAGE DUPLICATION .....	98
4.6.	XFDL EXERCISE 8 – DUPLICATING FORM PAGES .....	99
4.7.	WORKING WITH THE OUTLINE VIEW .....	107
4.7.1.	<i>What is the Outline View?</i> .....	107
4.7.2.	<i>How Do I Re-arrange the Build Order?</i> .....	107
4.7.3.	<i>How Do I Open the Outline View?</i> .....	107
4.7.4.	<i>What Is Filter On Visible?</i> .....	107
4.7.5.	<i>Copy/Pasting Objects in the Outline View</i> .....	107
4.7.6.	<i>Expanding/Collapsing View</i> .....	108
4.7.7.	<i>Page Navigating</i> .....	108
4.8.	RELATIVE ALIGNMENT .....	109
4.9.	XFDL EXERCISE 9 – IMPLEMENTING RELATIVE ALIGNMENT .....	111
4.10.	ITEM DUPLICATION .....	118
4.11.	XFDL EXERCISE 10 –DUPLICATING ROWS OF ITEMS .....	119
4.12.	XFDL EXERCISE 11 – CREATING MUTUALLY EXCLUSIVE CHECKBOXES .....	128

---

4.13.	CREATING SIGNATURE FILTERS .....	135
4.14.	XFDL EXERCISE 12 – CREATING SIGNATURE FILTERS.....	137
4.15.	XFDL EXERCISE 13 – IMPLEMENTING WEB SERVICES IN XFDL.....	143
<b>5.</b>	<b>BUILDING SIMPLE XFORMS FORMS .....</b>	<b>147</b>
5.1.	AUDIENCE .....	147
5.2.	OVERVIEW.....	147
5.3.	ADDING XFORMS SUPPORT – TEMPLATE VS. ENABLE .....	147
5.4.	CREATING XFORMS INSTANCES – VARIOUS METHODS.....	148
5.4.1.	<i>Creating a new empty instance</i> .....	148
5.4.2.	<i>Creating an instance from the current form</i> .....	149
5.4.3.	<i>Creating a new instance from a Web Service</i> .....	150
5.4.4.	<i>Creating a new instance from a schema</i> .....	150
5.5.	CREATING XFORMS OBJECTS.....	151
5.6.	CREATING ITEMS FROM AN XFORMS INSTANCE.....	153
5.7.	LINKS TO DATA MODEL.....	154
5.7.1.	<i>Highlighting Bound Items</i> .....	154
5.8.	XFORMS EXERCISE 1 – BUILD SIMPLE XFORMS FORMS .....	156
5.9.	TAB ORDER.....	168
5.10.	XFORMS EXERCISE 2 – MODIFYING THE TAB ORDER .....	170
<b>6.</b>	<b>UNDERSTANDING XPATH .....</b>	<b>171</b>
6.1.	AUDIENCE .....	171
6.2.	OVERVIEW.....	171
6.3.	NAMESPACES .....	172
6.4.	XPATH OPERATORS.....	173
6.4.1.	<i>Numeric</i> .....	173
6.4.2.	<i>Boolean</i> .....	173
6.5.	XPATH REFERENCES .....	174
6.5.1.	<i>Building XPath References</i> .....	174
6.5.2.	<i>The Root Location Path</i> .....	174
6.5.3.	<i>Locating Children Elements</i> .....	174
6.5.4.	<i>Sample XForms Instance</i> .....	175
6.5.5.	<i>Locating Attributes</i> .....	175
6.5.6.	<i>Predicates</i> .....	175
6.5.7.	<i>XPath Sample Form</i> .....	176
6.6.	XPATH FUNCTIONS .....	177
6.6.1.	<i>XPath Functions Sample Form</i> .....	178
<b>7.</b>	<b>ADVANCED XFORMS CONCEPTS .....</b>	<b>179</b>
7.1.	AUDIENCE .....	179
7.2.	OVERVIEW.....	179
7.3.	ADDING FORM LOGIC .....	179
7.4.	XFORMS BIND .....	179
7.4.1.	<i>readonly</i> .....	179
7.4.2.	<i>required</i> .....	180

7.4.3.	<i>relevant</i> .....	180
7.4.4.	<i>calculate</i> .....	180
7.4.5.	<i>constraint</i> .....	180
7.4.6.	<i>Creating a Bind</i> .....	180
7.5.	XFORMS EXERCISE 3 – IMPLEMENTING XFORMS BINDS.....	182
7.6.	XFORMS ACTIONS.....	189
7.7.	XFORMS EXERCISE 4 – USING XFORMS ACTIONS.....	190
7.8.	XFORMS EVENTS.....	196
7.9.	XFORMS EXERCISE 5 – USING XFORMS EVENTS.....	197
7.10.	XFORMS SUBMISSIONS.....	205
7.11.	XFORMS EXERCISE 6 – USING XFORMS SUBMISSIONS.....	206
7.12.	XFORMS TABLES.....	209
7.13.	XFORMS EXERCISE 7 – CREATING A TABLE (W/ TABLE WIZARD).....	210
7.14.	XFORMS EXERCISE 8 – CREATING A TABLE (W/OUT TABLE WIZARD).....	213
7.15.	XFORMS PANE.....	221
7.16.	XFORMS EXERCISE 9 – CREATE A MULTI-PANE SECTION.....	222
7.17.	WEB SERVICES AND SCHEMAS.....	226
7.18.	XFORMS EXERCISE 10 – IMPLEMENTING WEB SERVICES IN XFORMS.....	227
7.19.	XFORMS EXERCISE 11 – IMPLEMENTING SCHEMA VALIDATION.....	230
<b>8.</b>	<b>CHOOSING BETWEEN XFDL AND XFORMS.....</b>	<b>233</b>
8.1.	AUDIENCE.....	233
8.2.	OVERVIEW.....	233
8.3.	EXAMPLES IMPLEMENTING XFORMS VS. XFDL:.....	233
8.3.1.	<i>Item Visibility</i> .....	233
8.3.2.	<i>Item Status: Mandatory or Optional</i> .....	233
8.3.3.	<i>Item Value: Performing Simple Calculations</i> .....	234
<b>9.</b>	<b>APPENDIX A - LEARNING ECLIPSE COMPONENTS.....</b>	<b>235</b>
9.1.	WHAT IS A PERSPECTIVE?.....	235
9.1.1.	<i>How Do I Activate a Perspective?</i> .....	235
9.1.2.	<i>Customizing a Perspective</i> .....	236
9.1.3.	<i>Resetting a Perspective</i> .....	236
9.1.4.	<i>Closing a Perspective</i> .....	236
9.1.5.	<i>Saving a Customized Perspective</i> .....	236
9.2.	THE RESOURCE PERSPECTIVE.....	237
9.2.1.	<i>Managing Your Files</i> .....	237
9.3.	WHAT IS A VIEW?.....	240
9.3.1.	<i>How Do I Activate a View?</i> .....	240
9.4.	LOCAL HISTORY.....	240
9.4.1.	<i>What is the Local History?</i> .....	240
9.4.2.	<i>Accessing the Local History</i> .....	240
9.4.3.	<i>Comparing Your Current File Against the Local History?</i> .....	241
9.4.4.	<i>How Do I Replace a File With One From the Local History?</i> .....	241
9.5.	PROBLEMS VIEW.....	242
9.6.	TASKS VIEW.....	242

<b>NOTICES .....</b>	<b>243</b>
TRADEMARKS .....	244





# 1. Introduction

## 1.1. Purpose

The purpose of this document is to provide a detailed description of IBM's Workplace Forms Designer. The Designer is built on the Eclipse framework; this document describes how to use the Eclipse Interface in the context the Designer. It can be used as a reference guide to the Designer for building standard XFDL and XForms-enabled forms.

After reading this document, you should be able to:

- Understand the Difference between a Perspective and a View
- Recognize, Understand and Navigate the Designer Interface
- Create an XFDL form
  - Create XFDL items
  - Modify item properties
  - Use alignment tools to define your form layout
  - Change the tab order of your input items
  - Write simple computes
- Create an XForms form
  - Create XForms items
  - Create XML instances
  - Create links between your items and XML instances
  - Write simple XPath expressions

## 1.2. Audience

This is an introductory document to the Workplace Forms Designer, XFDL and XForms. It assumes that the audience has very little knowledge of these topics.



## 2. Building Simple XFDL Forms

### 2.1. Audience

This section is intended to teach someone the basic procedures for designing and creating an XFDL form in the Workplace Forms Designer. This module does not have any formal prerequisites. It is the basic introduction to the Workplace Forms Designer tool. It serves as the foundation for all other modules.

### 2.2. Overview

This module focuses on teaching the reader how to navigate the Designer's main windows and dialogs to design simple XFDL forms. After reading this section the user should:

- Be able to create any XFDL item on the designer palette
- Understand the differences between and use the selection types
- Be able to manipulate the properties of all the objects
- Be able to move/align items with the three Designer methods
- Be able to change the default tab order of a form
- Be able to enclose images within a form

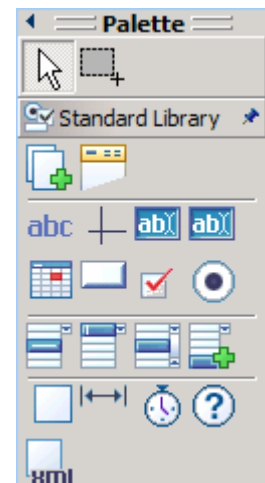
### 2.3. Working with the Palette

#### 2.3.1. What is the Palette?

The palette is the part of the workspace that contains all of the XFDL and XForms items that are used to build electronic forms.

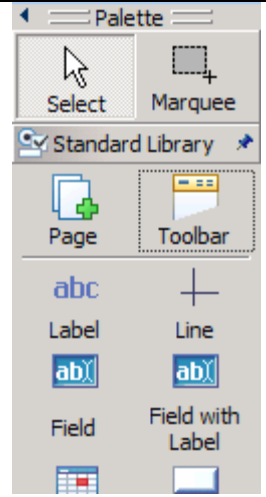
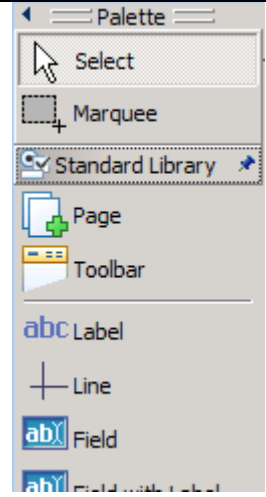
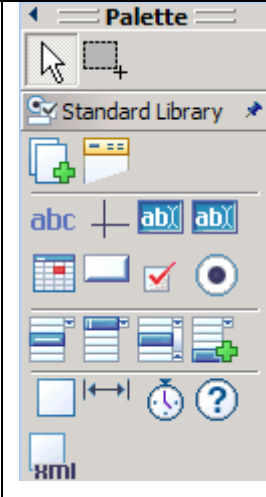
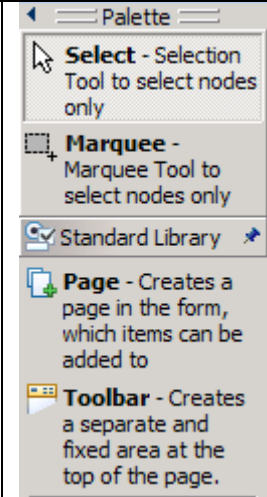
The palette is made up completely of buttons. Clicking an item button selects that object; if you click on the Designer canvas the selected item will be created. The drawer buttons are toggles that open and close the drawers.

The palette is a separate window that can be placed on the left or right side of the canvas (depending on your preference). To move the Palette click and hold the left mouse button over the window header (the section of the window that contains the text "Palette") and drag it to the desired location. As you drag the window the outline will snap to a valid location (either left or right side of the canvas).



### 2.3.2. Palette Layout

The palette is responsible for displaying all the items that can be placed on an XFDL form. There are several different ways in which those items can be displayed:

Columns	List	Icons Only	Details
			

#### Large Icons

The picture included in section 2.3.1 is using the Large Icons option, which can be enabled by right-clicking on the palette and choosing “Use Large Icons”.

### 2.3.3. Selection Types

There are a few different selection types that are available to you as you are working your way around the Designer.

#### Marquee



Marquee is the term that describes the process of selecting items by drawing a selection box around them. To enable the marquee tool press the corresponding button on the palette.

#### Select



The select type is the default selection tool; an item is selected by clicking it. Multiple items can be selected by holding down ctrl and clicking the desired items.

#### Connection Creator



The connection creator is the tool that allows you to define the tab order of the form input items. Once the tab mode has been activated (please see section 2.11 for details about activating tab mode) the connection tool will appear in the palette. You create relationships between items by clicking the first object and then the second object, a line with an arrow indicating direction will be drawn between the two items.

### 2.3.4. Creating Palette Drawers

#### What Are Drawers?

The Eclipse palette is made up of drawers of items. All of the XFDL and XForms items have been placed into a drawer called “Standard Library” (The XForms items will only appear if XForms support has been enabled). The drawers can be collapsed or expanded to hide or show their contents.



























	Drawer Collapsed
	Drawer Expanded

#### How Do I Create My Own Drawers?

Additional drawers can be created to contain custom objects (groupings of multiple XFDL or XForms items). The custom drawer will get created when you export your custom items. Refer to section 2.3.5 XFDL Objects for more information on exporting objects.

### 2.3.5. XFDL Objects

For complete item definitions or specific implementation details please refer to the XFDL manual.

Icon	Item	Icon	Item
	Action		Help
	Box		Label
	Button		Line
	Cell		List
	Check		Page
	Check Group		Popup
	Check Group Item		Radio button
	Combobox		Radio Group
	Date Picker		Radio Group Item
	Field		Signature
	Field (Text Area)		Spacer
	Field (Secret)		Toolbar
	Field with Label		Custom Item

#### How do I create Custom Objects?

1. You will want to have a project that is dedicated to your custom objects. Let's create this project first.
  - a. Select **File > New > Project**
  - b. Click **Next >**
  - c. Enter the name of your project (i.e. "My Object Library")
  - d. Click **Finish**
2. Select the item or groups of items that you want to be saved as a custom object, right-click on one of the selected objects and select **Export Objects....**
3. Set the **Form Object Library Path**.

- a. Click the **Add Path** button and browse to the folder that will contain all your custom objects (we created this in a previous step, i.e. "My Object Library").
4. Set the **Form Object File Name**.
  - a. Enter the name of the file that will be used to store the XFDL code for the custom object
5. Set the **Form Object Palette Name**.
  - a. Enter the name that will be displayed on the palette
6. Set the **Palette Drawer Name**.
  - a. Enter the name of the drawer (i.e. "My Objects"). If you have already created a drawer then you can select it from the popup list.
7. Click **OK**, now the custom object should appear as part of the palette and can be used on other forms.

**How do I remove custom drawers?**

1. Open the **Windows** menu and select **Preferences...**
2. Expand **Workplace Forms** and select **Form Object Library**
3. Select the object library you want to remove and click the **Remove** button

**2.3.6. XFDL Object Library**

We have created an object library that contains several common form items. These items can be used just like any of the standard XFDL items.

CDN Address Block	<table border="1" style="width: 100%;"> <tr> <td>First Name</td> <td>Middle Name</td> <td>Last Name</td> <td>S.I.N.</td> </tr> <tr> <td colspan="3"></td> <td>- -</td> </tr> <tr> <td colspan="4">Street Address</td> </tr> <tr> <td>City</td> <td>Province</td> <td colspan="2">Postal Code</td> </tr> <tr> <td colspan="2"></td> <td>Select Province</td> <td>▼</td> </tr> <tr> <td>Gender</td> <td>Date of Birth</td> <td>Work Phone</td> <td>Home Phone</td> </tr> <tr> <td></td> <td></td> <td>( ) -</td> <td>( ) -</td> </tr> </table>	First Name	Middle Name	Last Name	S.I.N.				- -	Street Address				City	Province	Postal Code				Select Province	▼	Gender	Date of Birth	Work Phone	Home Phone			( ) -	( ) -
	First Name	Middle Name	Last Name	S.I.N.																									
				- -																									
	Street Address																												
City	Province	Postal Code																											
		Select Province	▼																										
Gender	Date of Birth	Work Phone	Home Phone																										
		( ) -	( ) -																										
Phone Number (###) ###-####	Phone: ( ) -																												
Phone Number w/ Patterns	Phone:																												
Postal Code	Postal Code:																												
State Popup	State: Select State ▼																												

## 8 Building Simple XFDL Forms

State Popup (Abbreviated)	State: <input type="text"/> ▼																																
Province Popup	Province: <input type="text" value="Select Province"/> ▼																																
Province Popup (Abbreviated)	Province: <input type="text"/> ▼																																
SIN (###-###-###)	SIN: <input type="text" value="- -"/>																																
SSN (###-##-####)	SSN: <input type="text" value="- -"/>																																
Zip Code	Zip Code: <input type="text"/>																																
US Address Block	<table border="1"> <tr> <td>First Name</td> <td>Middle Name</td> <td>Last Name</td> <td>S.S.N.</td> </tr> <tr> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text" value="- -"/></td> </tr> <tr> <td colspan="4">Street Address</td> </tr> <tr> <td colspan="4"><input type="text"/></td> </tr> <tr> <td>City</td> <td>State</td> <td colspan="2">Zip</td> </tr> <tr> <td><input type="text"/></td> <td>Select State ▼</td> <td colspan="2"><input type="text"/></td> </tr> <tr> <td>Gender</td> <td>Date of Birth</td> <td>Work Phone</td> <td>Home Phone</td> </tr> <tr> <td><input type="text"/></td> <td><input type="text"/></td> <td>( ) -</td> <td>( ) -</td> </tr> </table>	First Name	Middle Name	Last Name	S.S.N.	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="- -"/>	Street Address				<input type="text"/>				City	State	Zip		<input type="text"/>	Select State ▼	<input type="text"/>		Gender	Date of Birth	Work Phone	Home Phone	<input type="text"/>	<input type="text"/>	( ) -	( ) -
First Name	Middle Name	Last Name	S.S.N.																														
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="- -"/>																														
Street Address																																	
<input type="text"/>																																	
City	State	Zip																															
<input type="text"/>	Select State ▼	<input type="text"/>																															
Gender	Date of Birth	Work Phone	Home Phone																														
<input type="text"/>	<input type="text"/>	( ) -	( ) -																														



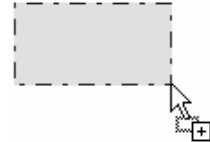
## 2.4. Working with the Canvas

When in the GUI Editor View, the canvas is the white space next to the palette where the form is built by placing, moving and manipulating items.

There are two ways to create objects on the canvas; you can choose to use the default size that has been assigned to every object or define your own. If you want to use the default size of an object, select the desired item tool from the palette and then click (the left mouse button) on the canvas.



If you want to define the size of the item being placed on the canvas, select the item tool from the palette, click (and hold the left mouse button) on the canvas and drag the bounding box (represents the item to be created) to the desired width and height.



The top left corner of the newly created item will be aligned with the coordinates of the initiating mouse click.

### Why Does it Matter?

Deciding when to use the different creation techniques is important and can greatly decrease the amount of time required to build a form. Remember that presentation can make the difference between a great form and one that is unusable.

By default, when width and height arguments are not specified for an item, the size will be determined by its value. A label's width will be determined by the length of the text that you provide, the width of a popup will be determined by the length of its longest option, etc. When the size of an item is defined by dragging the mouse the height and width are written into the XFDL code. The items will no longer expand based on their data; they are now locked to their set dimensions.

This label will grow as I add text to it.

This label will not grow be

Label was created with default size

Label was stretched manually

When creating items choose the default sizing, that way they will start with the same dimensions and save you from having to adjust the height or width later on in the form development process.

## 2.5. Working with Item Properties

Every item has properties that impact the visible or operational state of the object. The properties window is where all of those properties are listed.

The list is sorted into categories

Only the most common properties are shown by default. Access this menu to show all of them

Indicates the value has been modified

Property	Value
<b>General</b>	
sid	◦ title_FIELD
value	● ◦
label	◦ ◦ Title:
active	◦ ✓ on
previous	◦ ◦
next	◦ ◦
readonly	◦ ✗ off
texttype	◦ ◦ text/plain
rtf	◦ ◦
scrollhoriz	● ON wordwrap
scrollvert	● ON fixed
+ itemlocation	● x=30,y=60
<b>Appearance</b>	
justify	◦ left
border	◦ ✓ on
visible	◦ ✓ on
+ fontinfo	● T 8 pt Helvetica (plain)
+ fontcolor	◦ black
+ bgcolor	◦ white
<b>Format</b>	
<b>Help</b>	

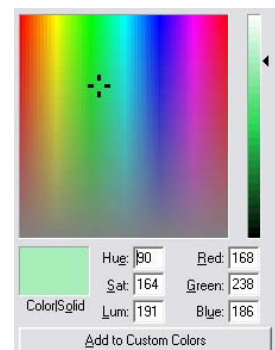
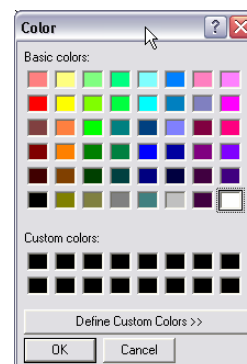
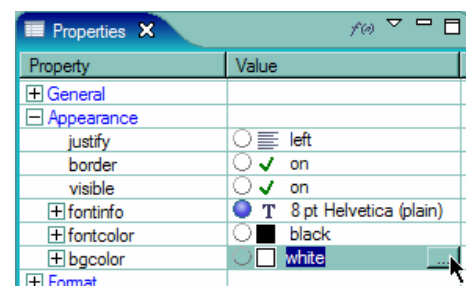
### 2.5.1. Changing a property

#### Changing a color

Colors are a common property of many XFDL items, from textboxes to pages; nearly every XFDL object can be assigned a color for at least one of its properties. The properties window provides a really simple mechanism for choosing your preferred color.

Select the color you want to change and then click on the button with the three dots (“...”). A color dialog will appear that allows the user to choose from a defined color set.

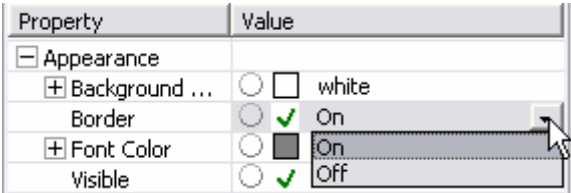
If the set does not contain the color you want, clicking on “Define Custom Colors >>” will expand the window allowing you to create your own. You can either specify the numerical values for: Red, Green and



Blue, the Hue, Saturation, and Luminescence, or you can find one in the color wheel.

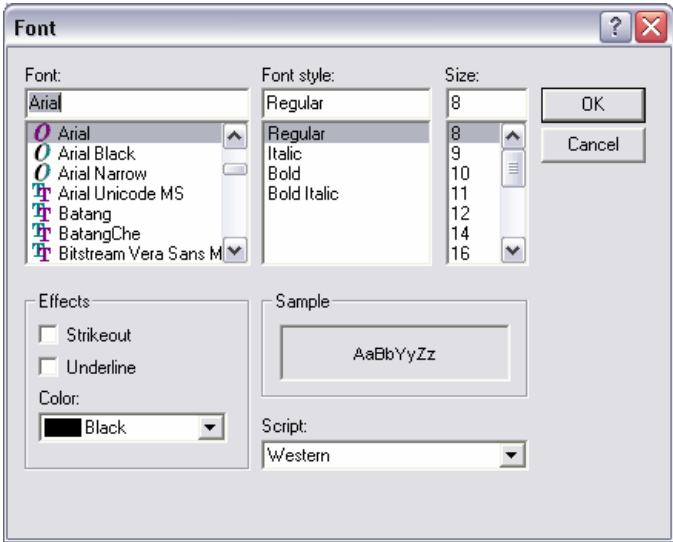
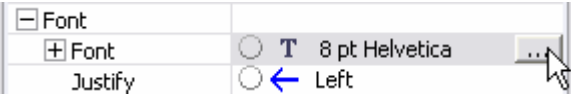
**Changing a toggle (on/off)**

Several of the properties are toggles that can be set to either “on” or “off”. To set this property type, select the option from the drop down.



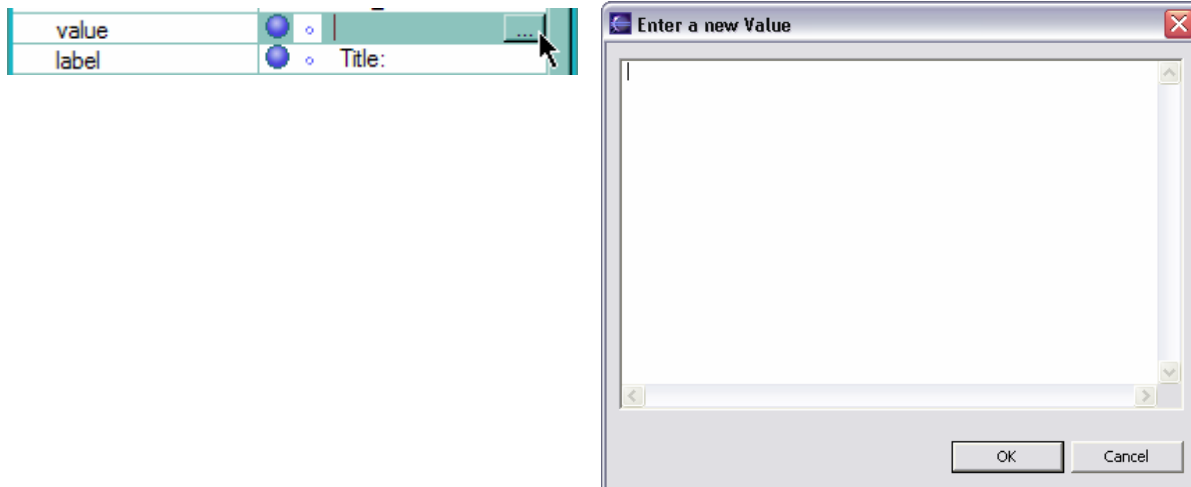
**Changing a font**

For changing the font attribute, the Designer provides a button that will open a new dialog.



### Changing a value (text options)

When changing a text property the user can enter it into the text field (in the properties window) or if they require additional space can launch a text entry dialog by pressing the button with the three dots (“...”).



### Adding a Compute

*f(2)*

Clicking on the “...” button will initialize the compute wizard.



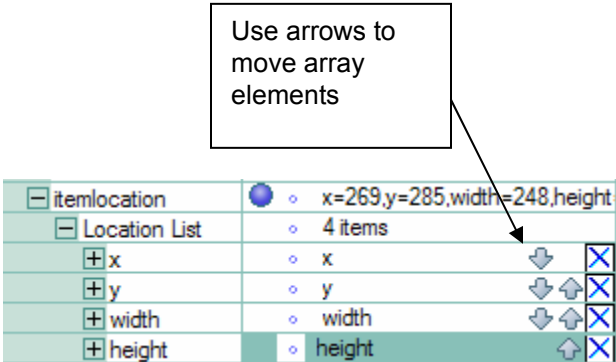
For a more detailed discussion of computes please refer to section 3.8 Computes.

**Changing sub properties**

Some properties contain sub-properties; sub properties are also known as arguments (or array elements). These sub properties can be manipulated as indicated below.

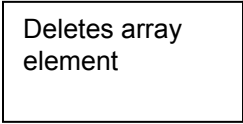
**Re-ordering sub properties**

Sub properties, or array elements, can be re-ordered at any time. The up and down arrows allow the user to change an elements position in an array. This is particularly useful when working with the itemlocation array since the elements are executed and evaluated in order from 0 to the length of the array.



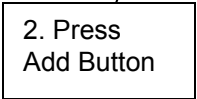
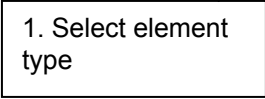
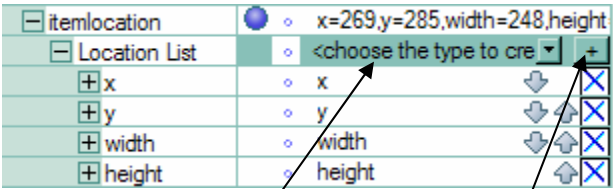
**Removing sub properties**

The array elements can be removed from an array by pressing the corresponding delete button (i.e. square that contains an "X").



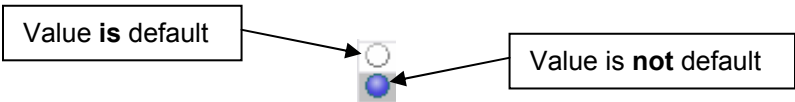
**Adding sub properties**

New array elements can be added to an array. First select the root node, in this case it is the "Location List". A drop down and button will appear. Select the type to be added (i.e. x, y, alignl2l, etc.) and press the add button.



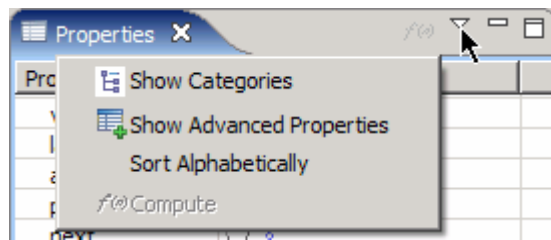
**2.5.2. Resetting a property back to the default value**

A property's value can be set back to the default value by clicking on the small toggle button. The value is written into the XFDL code if the toggle is on (circle is filled), otherwise the default value is assumed and it may not appear in the XFDL code.



### 2.5.3. Filtering the Properties View

The Properties View contains a few filters that change the way the properties are presented.



#### Show/Hide Categories

All the properties have been assigned categories. You may show or hide the category headings.

#### Show/Hide Advanced Properties?

Show or hide properties that are used less frequently.

#### Sort Alphabetically

Sort the list of properties alphabetically. If Show Categories is on then the properties will be sorted relative to their categories, otherwise all the properties will be listed in order from a-z.

## 2.6. Form Preview

The Preview shows what the form looks like in the Viewer. When you press the Preview button the Viewer will be embedded within Eclipse. Forms that are previewed in Eclipse may have a few toolbar functions disabled; for example the forms cannot be emailed.

## 2.7. XFDL Exercise 1 - Build Simple XFDL Form

The purpose of this exercise is to teach the reader how to navigate the Designer in order to build simple XFDL forms. Each of the XFDL widgets will be created and manipulated.

At the end of the exercise your form should look like the picture on the right. Keep this in mind as you are following the steps and creating your objects.

### Getting Started

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 > Designer**
2. Open all the training projects (i.e. **TrainingExercises\_XFDL**, **TrainingExercises\_XForms**, **TrainingSamples\_XForms**)
  - a. **File > Import > Existing Projects into Workspace** and click **Next >**.
  - b. Browse to the location of the existing projects and click **Finish**.
3. There are two perspectives that are available by default; the Designer and Resource. The Designer perspective defines the basic views that will be required to build a form. The Resource perspective defines the views that are required to manage your projects and files.
  - a. Open the **Resource** perspective by selecting **Window > Open Perspective > Other...**, choose Resource from the new dialog and then click **OK**.
  - b. Look for your project in the **Navigator View**.
4. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. "simpleXFDLForm"), then click **Finish**
  - c. The Designer should automatically switch to the Designer Perspective and open the file.

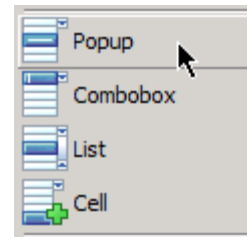
XFDL Items

<p><b>Attach</b></p> <p>Display Attachments</p> <p style="background-color: blue; color: white; text-align: center; padding: 2px;">[Redacted]</p> <p>Date of Birth</p> <p><input type="checkbox"/></p>	<p>What is your Favorite Color? ▼</p> <p>What is your favorite Italian Dish?</p> <p>What is your favorite fruit?</p> <p>▼</p> <p>What is your favorite bird of prey?</p> <p>Eagles    Hawks    Falcons</p> <p><input type="radio"/>    <input type="radio"/>    <input type="radio"/></p>								
<p>What is your favorite greek dish?</p> <table style="width: 100%; border: none;"> <tr> <td style="border: none;">Moussaka</td> <td style="border: none;">Pastitsioto</td> </tr> <tr> <td style="border: none;"><input type="checkbox"/></td> <td style="border: none;"><input type="checkbox"/></td> </tr> <tr> <td style="border: none;">Souvlaki</td> <td style="border: none;">Gyros</td> </tr> <tr> <td style="border: none;"><input type="checkbox"/></td> <td style="border: none;"><input type="checkbox"/></td> </tr> </table>	Moussaka	Pastitsioto	<input type="checkbox"/>	<input type="checkbox"/>	Souvlaki	Gyros	<input type="checkbox"/>	<input type="checkbox"/>	
Moussaka	Pastitsioto								
<input type="checkbox"/>	<input type="checkbox"/>								
Souvlaki	Gyros								
<input type="checkbox"/>	<input type="checkbox"/>								

### Add a popup item to the form



5. Click on the popup item from the Standard Library on the palette and click on the canvas to create the popup.
6. The group property is used to associate choices with the popup. Each list must be assigned a group, however multiple lists may share the same group if the choices are the same between the lists.

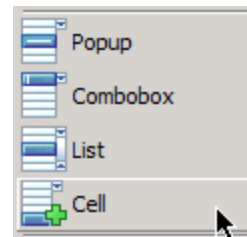


- a. In the **Properties View**, set the popup's **group** to 'color\_group'.
7. The label property in the **Properties View** sets the label of the popup item when the form is viewed. A popup label appears as the default selection; once a selection has been made there is no way to get back to the label.
  - a. Set the value of the **label** to 'What is your Favorite Color?'

Property	Value
[-] General	
sid	o POPUP1
value	o
group	o POPUP1_GROUP
label	o What is your Favorite Color?
active	<input type="checkbox"/> on

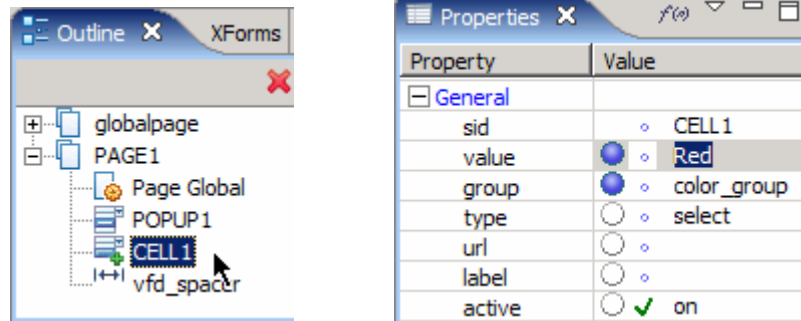
### Add cells (Options) to the popup

8. The selections within a list have been defined as their own XFDL item, they are called cells. One reason for making list selections independent of the main list item is so that they can be re-used between multiple lists (popups, combo boxes and list boxes) within a single form. Another advantage of having cells is that they can store their own individual properties, for instance "type". Let's add some cells to our popup that we just created.



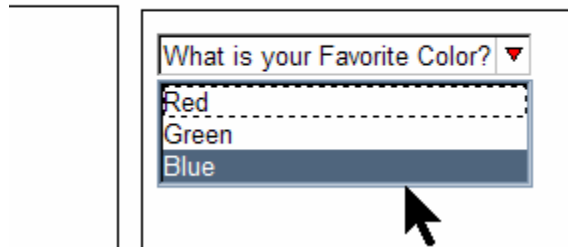
- a. Left click the **Cell** item from the Standard Library on the palette and then left click on the popup item on the canvas to create the cell. The cell is automatically assigned to the selected popup's group.
9. Notice that the cell is invisible on the canvas, but if you look in the Outline View you will see that an object has been created with a SID of "CELL1". If you want to modify the cell object you must find it and select it in the Outline View. Once the cell is selected you can then modify its properties in the Property View.
  - a. Find **CELL1** and set its value to **Red**.





10. Add two more cells with values 'Green' and 'Blue'.
11. Preview the form in the Preview Mode. Try making a selection in the popup, notice that the three colors that you added (in the form of cells) are included within the popup list.

## XFDL Items



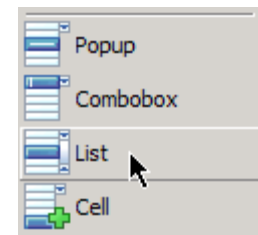
### Save Your Form

12. Now is a good time to save your form. Get into the habit of saving your form often. Every time that you save your form Eclipse adds a record to the local history that can then be reviewed or restored; it acts a local versioning system.
  - a. Save your form by selecting **File > Save**, or by pressing **Ctrl + S**.

### Add a list item to the form



13. Select the **List** item from the Standard Library on the palette. Left click on the canvas to place the list on the form.



14. The default size for the list is quite small so expand the object by selecting one of the corners and dragging outward.



15. Change the value of the **group** property to “**italian\_dish\_group**” and set the label to ‘**What is your favorite Italian Dish?**’

Property	Value
General	
sid	LIST1
value	
group	italian_dish_group
label	What is your favorite Italian Dish?
active	on

16. Next we will add a few cells to the List.



- a. Left click the **Cell** item from the Standard Library on the palette and then left click on the list item (on the canvas). The cell is automatically added to the list's group.
- b. Find the new cell in the **Outline View** and change its **value** to ‘**Pizza**’.

Property	Value
General	
sid	CELL2
value	Pizza
group	italian_dish_group
type	select

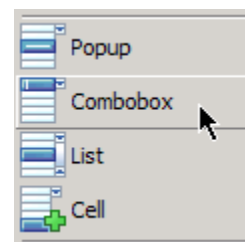
17. Add three more cells with values ‘**Lasagna**’, ‘**Spaghetti**’ and ‘**Calazone**’.

18. Preview the form in the Preview mode.

## XFDL Items

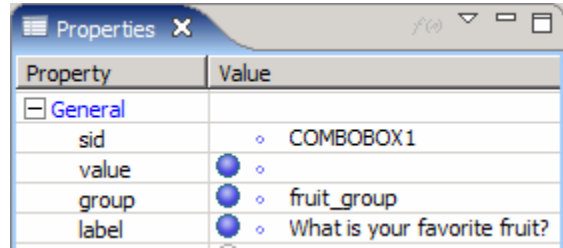
### Add a combobox item to the form

19. Select the combobox item from the Standard Library on the palette and left click on the canvas to create the combobox.

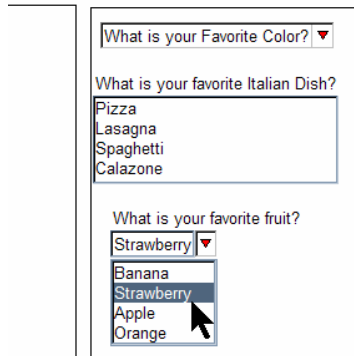


20. Resize the Combobox so that it looks like the one in the screenshot of the example (that was provided at the beginning of this exercise).

21. Set the value of the **group** property to **'fruit\_group'** and the value of the **label** property to **'What is your favorite fruit?'**
22. Add four cells to the combo box with values **'Banana', 'Strawberry', 'Apple'** and **'Orange'**.
23. Preview the form.

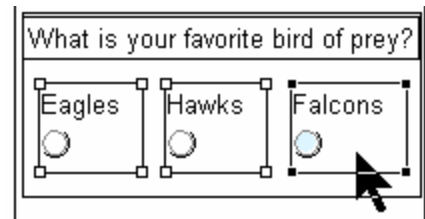


### XFDL Items

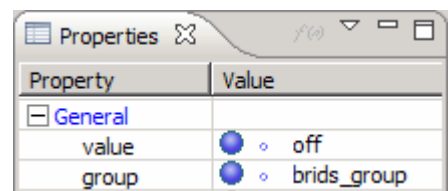


### Add a group of radio buttons to the form

- abc 24. Add a **label** to the form with value set to **"What is your favorite bird of prey?"**
25. Select the radio item from the Standard Library on the palette.
  - a. Left click on the canvas to place the radio button on the form.
  - b. Change the value of the **label** property of the radio button to **"Eagles"**.
26. Add two more radio buttons with labels set to **"Hawks"** and **"Falcons"**.
27. Select all the radio buttons
  - a. Left click each radio button while holding down the control key or by clicking the canvas and dragging a bounding box around each item while the mouse button is held down (all the items in the bounding box when the mouse button is released will be selected).



- b. Change the value of the **group** property to **"birds\_group"**. Doing this associates all the radio buttons to the same group since radio buttons are mutually exclusive, in preview you will see that selecting one will turn the others off.



- c. Preview the form.

**XFDL Items**

What is your Favorite Color? ▼

What is your favorite Italian Dish?

Pizza  
Lasagna  
Spaghetti  
Calazone

What is your favorite fruit?

Strawberry ▼

What is your favorite bird of prey?

Eagles     Hawks     Falcons

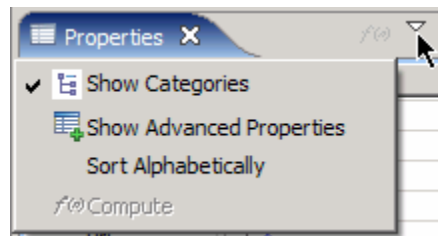
### Create an Add Attachment button to the form

28. Select the **Button** item from the Standard Library on the palette. Left click on the canvas to place the button on the form.

- a. Change the **value** property of the button to “**Attach**” and set its **type** to **Enclose**.

Property	Value
<b>General</b>	
sid	o BUTTON1
value	o Attach
type	o enclose

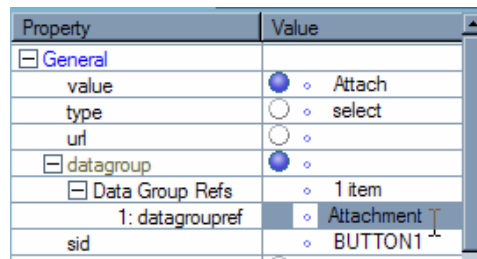
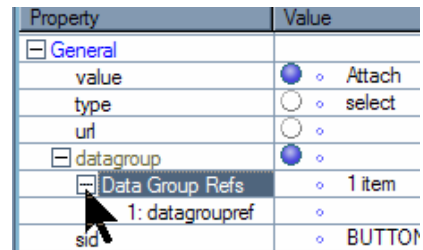
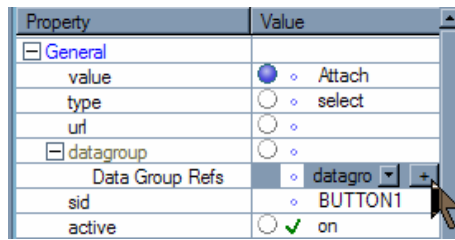
- b. Left click on the down arrow in the properties dock and click on “Show Advanced Properties”



- c. Add a **datagroup** to the button. The datagroup option identifies the list of datagroups, or folders, in which the user can store the enclosed file.

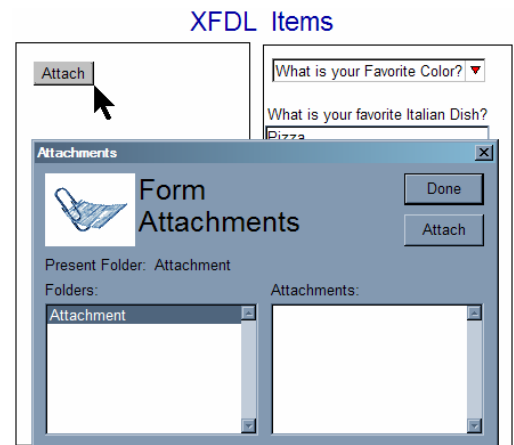
Note: All attachments added to a form must be placed within a datagroup.

- d. Click on the '+' button to add a datagroupref to the datagroup and set the **value** to '**Attachment**'. We are using the string 'Attachment' but this can be any string; you might want to give it a name that depicts the type of attachments to be added. You can also add multiple datagroups if you want your users to add multiple attachments and place them into separate logical groups.



- e. Preview the form.

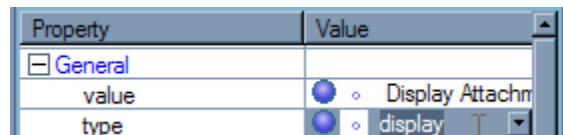
- i. In the Viewer attachments are added by clicking the "Attach" button. A new dialog will appear.
- ii. Click the **Attach** button to bring up an open dialog, which you use to navigate your drive and add a file to the form as an attachment.



**Create a Display Attachment button to the form**

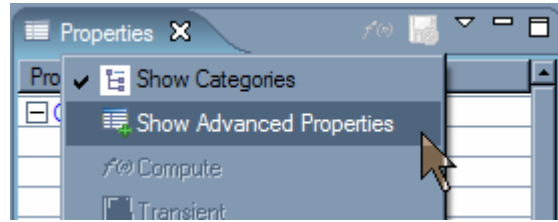
- 29. Select the button item from the Standard Library on the palette. Left click on the canvas to place the button on the form.

- a. Change the value of the button to "Display Attachments". Set the type to

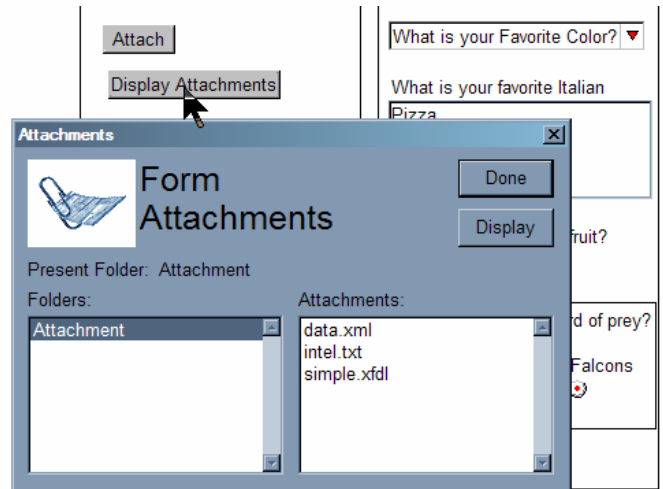


Display.


- b. Left click on the down arrow in the **Properties View** and click on **“Show Advanced Properties”**. If you created the previous item then this step should have already been completed.



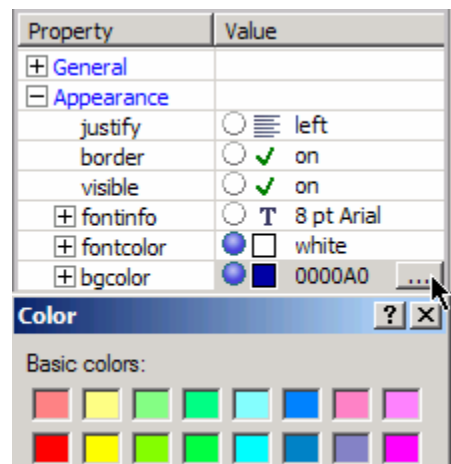
- c. Add a datagroup to the button.
  - i. The datagroup option identifies the list of datagroups, or folders, in which the user can display.
  - ii. Click on the '+' button to add a datagroupref to the datagroup and set the value 'Attachment'.
- d. Preview the form.
  - i. Attach some files and then click on the display button. A dialog will appear that lists all the files currently attached to the form.
  - ii. If you select an attachment and click Display, the Viewer will invoke the parent application to display the selected attachment (i.e. a .doc file will open in Word, an .xpdf file will open in a new instance of the Viewer, etc).



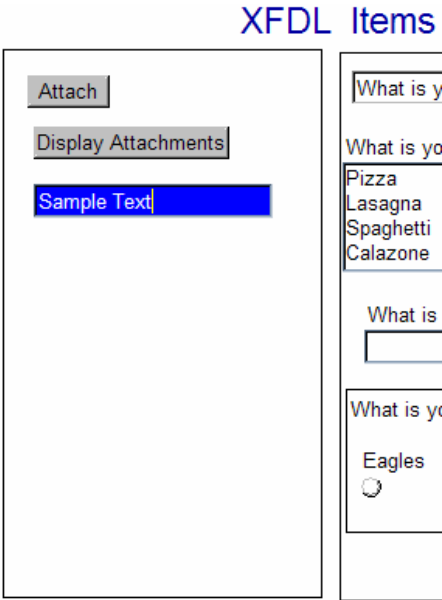
**Create a field**

 30. Select the field from the Standard Library on the palette. Left click on the canvas to place the field on the form.

31. Change the background color (**bgcolor**) to dark blue as shown below and the text color (**fontcolor**) to white.



a. Preview the form.

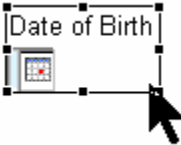


**Add a date picker**

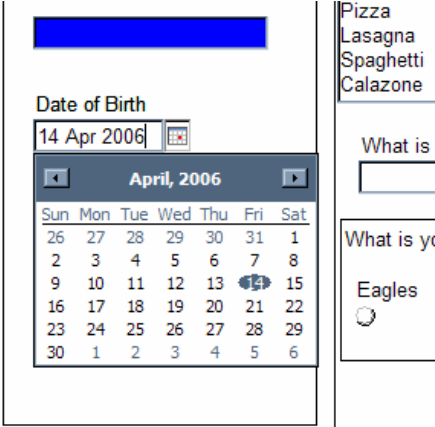
32. Select the **Date Picker** from the Standard Library on the palette. Left click on the canvas to place the Date Picker on the form.

a. Set the value of the **label** property to "Date of Birth".

Property	Value
group	<input type="radio"/>
label	Date of Birth
sid	COMBOBOX2

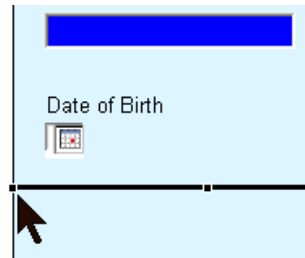


b. Preview the form.



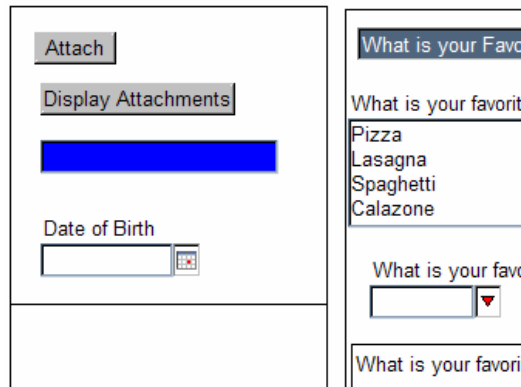
**Add a line**

- 33. Select **Line** from the Standard Library on the palette. Left click on the canvas to place the Line on the form as shown in the picture.



- 34. Preview the form.

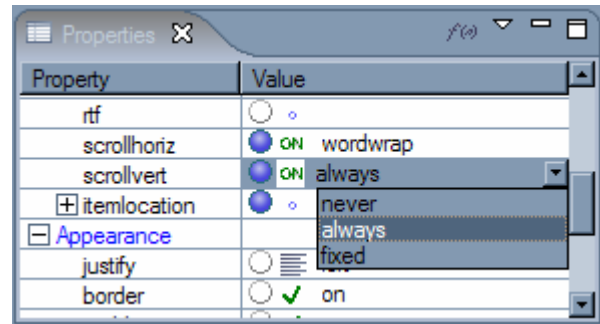
**XFDL Items**



**Add a Text Area**

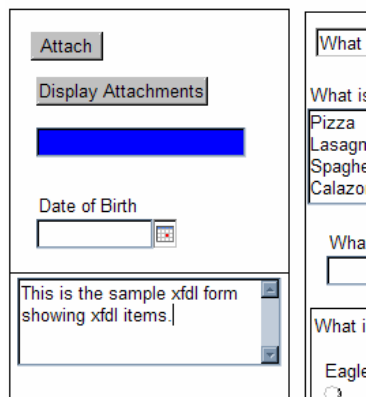
- 35. Select the Field from the Standard Library on the palette. Left click on the canvas to place the field on the form.

- a. Change the value of the **scrollvert** property to **always**. This will add vertical scrollbars to the field.
- b. Resize the object so that it is large enough to accept multiple lines of text.



- c. Preview the form.

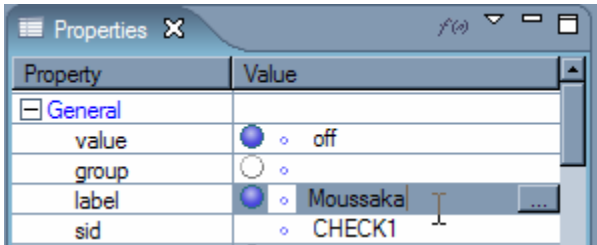
**XFDL Item**





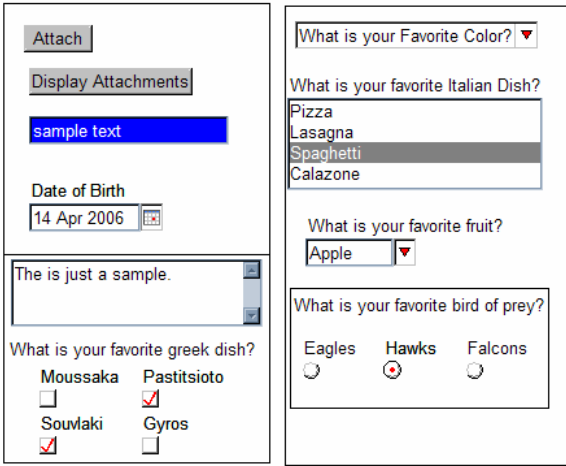
**Add Checkboxes**

- abc 36. Select the **Label** item from the Standard Library on the palette and left click on the canvas to create the label.
  - a. Change the **value** of the label to **'What is your favorite Greek dish?'**
- ✓ 37. Select the **Check** item from the Standard Library on the palette and left click on the canvas to create a checkbox. Change the value of the **label** property to **'Moussaka'**.



- 38. Add three more checkboxes with labels set to **Pastitsioto**, **Souvlaki** and **Gyros**.
- 39. Preview the form.

**XFDL Items**



## 2.8. Grid Options

Some people find that using a grid is helpful when designing a form's layout. The Workplace Forms Designer supports several grid related options

### Show Grid

The grid lines can be enabled from the File Menu, select **View > Show Grid**.

### Snap To Grid

If you want the grid lines to serve as more than just a visible marker, then you can enable "Snap To Grid". This feature will cause items to "snap" to the grid lines when being moved or resized.

Snap To Grid can be enabled from the File Menu, select **View > Snap To Grid**.

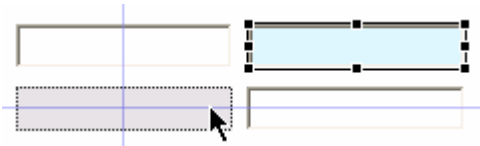
### Adjusting Grid Options

The settings for the grid can be adjusted to suit your development style. The settings can be found by selecting **Window > Preferences**. Expand Workplace Forms and Design View to find the options for setting the x and y coordinate spacing and the grid style (choice between lines and dots).

Expand Workplace Forms, Design View and Colors to change the grid color.

### Snap To Geometry

Snap To Geometry is similar to snap to grid except instead of the objects "snapping" to the grid lines they snap to other objects on the canvas. When moving an object additional guide lines will appear when its edge is inline with another item on the form. The edges that can be used as "snapping" points are the top, left, right, bottom, and center.



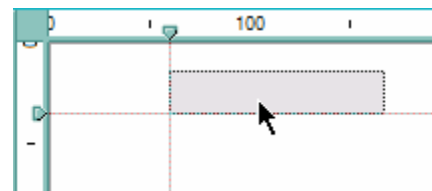
Snap To Geometry can be enabled from the File Menu, select **View > Snap To Geometry**

### Show Rulers

The Canvas can be enhanced by adding rulers to the top and left edge. The default measurement is pixels. To activate the Rulers select **View > Show Rulers** from the File Menu.

### Creating Guides

A Guide is a line that is connected to the ruler and stretches the length of the canvas. Guide lines are helpful for working on the form layout because when you move an item near the line it will snap to its edge. Guide lines can be created both vertically and horizontally.



To create a guide line the Rulers must first be active. Click on the ruler in the location where you want the guide line to appear. Once a guide line has been created it can be moved along the ruler.

### **Removing Guides**

Guide lines are not visible in the Viewer, however there may come a time where you want to remove the guide lines that you created to aid in its design. Every guide line has a mark that appears on the ruler indicating its pixel location. A guide line is removed by clicking on the ruler mark and dragging it off the ruler.

### **Adjusting Ruler Units**

The default measurement is pixels. There is no mechanism for adjusting the ruler unit in this version.

### **Showing the Page Size Marker**

All forms usually have a page size. To aid the form designer in creating the layout for their form within the confines of the page the Workplace Forms Designer has a Page Size Marker. The marker shows the outline of the current page size so that you can place your items appropriately.

To activate the page size marker select **View > Show Page Size** from the File Menu.

## 2.9. Refining Form Layout

### 2.9.1. Moving Objects

Moving objects that have already been placed on the canvas is a big part of form design. Rarely does one have enough foresight to place each item exactly where they want it the first time, especially as the form grows in size and complexity. There are two different mechanisms that have been provided that enable a form designer to move the objects around the canvas.

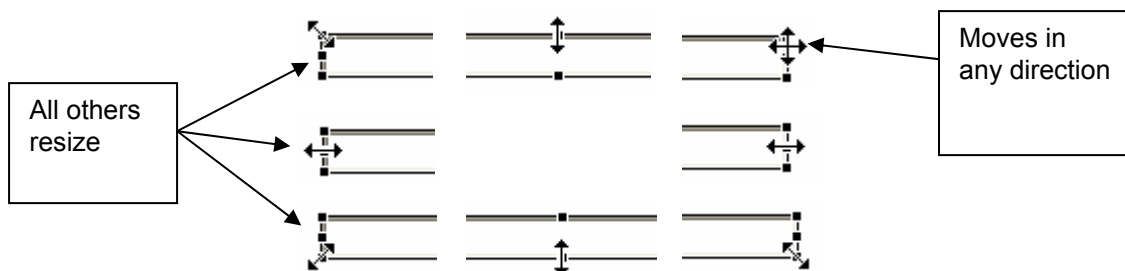
#### Manually

Every item on the canvas can be moved manually, individually or grouped with other items. Select the item(s) to move, and drag the item(s) to a new location on the canvas. This method of moving objects is crude at best, since you are relying on your ability to “eyeball” the items into place.

The Shift key can be used to constrain the movement of the object (or group of objects) to the first direction that the mouse is moved (either vertical or horizontal)

#### Arrow Keys

The arrow keys can also be used to “nudge” or “expand” an item by 1 pixel in any direction. Pressing the period key (“.”) will enable this functionality; it moves the focus (cursor) around the main indices of the selected item clockwise. The arrow keys will modify the item according to the rules of the modifier that is visible. Holding shift while pressing the period key will cause the cursor to move around the item counter-clockwise.



#### Position Modifiers

The more complex a form the finer the control required to align and move objects. There are several different movement modifiers that provide the ability to quickly align or expand multiple objects at the same time. These modifiers have been divided into two main groups (Relative and Absolute) and two sub groups (Align and Expand), which coincide directly with the XFDL itemlocation concepts. For more information about how to form proper relative or absolute itemlocations please refer to the XFDL manual.

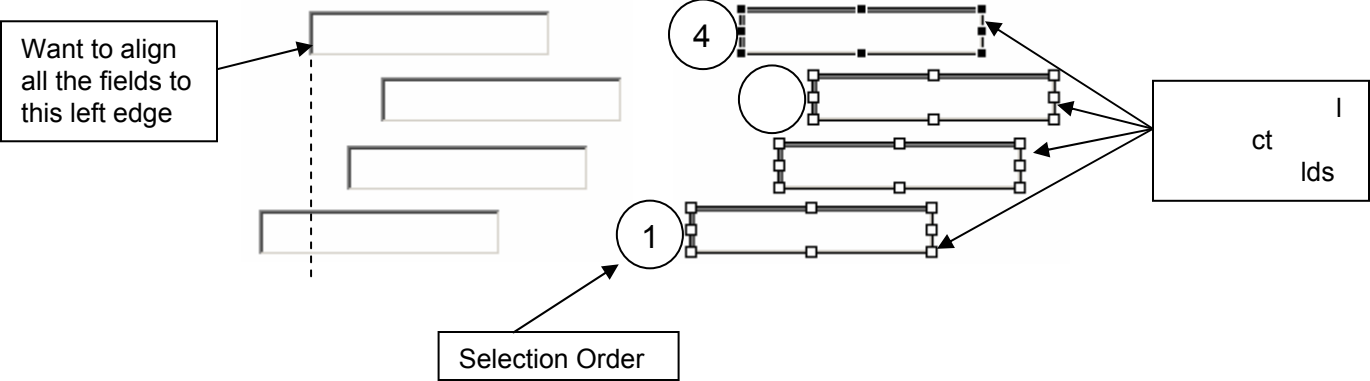
All of the position modifiers are implemented using a similar procedure. When an object is selected it is surrounded by a bounding box with black dots on each index and at the center point of each side.



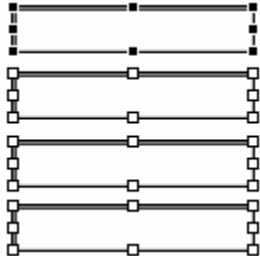
As more items are added to a selection (by holding the shift or ctrl key) the item that has the black dots is the reference object. The last item selected is always the reference object (and therefore is surrounded by the black dots). The reference object can be changed by holding the shift key and clicking an item in the selection group.



The position modifiers apply a particular operation to a selection of multiple items.



Right-click on the last object selected, choose the position modifier (in this case “Align Left To Left” which can be found under “Absolute Align”). All the items will snap directly beneath the reference object.



Even though the position modifiers all perform different actions, this procedure can be applied to all of them. Let’s re-cap it:

1. Select the objects to be manipulated. Make sure that your reference object is the last one selected (indicated by black dots).
2. Right-click, choose the position modifier.

**2.9.2. Aligning Objects**

**Align Modifiers**

Category	Modifier
Top Edge	
	Align Top to Top
	Align Top to Bottom
	Align Top to Center
Left Edge	
	Align Left to Center

	Align Left to Left
	Align Left to Right
Bottom Edge	
	Align Bottom to Center
	Align Bottom to Top
	Align Bottom to Bottom
Right Edge	
	Align Right to Center
	Align Right to Left
	Align Right to Right
Center	
	Align Center to Bottom
	Align Center to Left
	Align Center to Right
	Align Center to Top
	Align Center to Center Vertically
	Align Center to Center Horizontally
Reposition	
	Align Above
	Align Below
	Align After
	Align Before
Axis Spacing	
	Align Horizontally Between
	Align Vertically Between
	Horizontally equal spacing
	Vertically equal spacing

**Absolute Align**

Absolute alignment means that an item’s location will be recorded as **absolute** coordinates; the x, y, width and height values will be defined as pixels.

**Relative Align**

Relative alignment means that an item’s location is defined in **relation** to another item. Therefore when an item is moved all other items whose location is relative to the first will move as well.

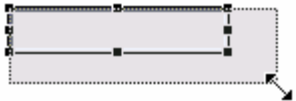
**2.9.3. Expanding Objects**

**Manually**

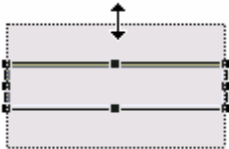
When an item is selected eight nodes appear which form the “bounding box”. All items can be expanded manually using the bounding box. Each of the nodes can be selected and moved to manipulate the object contained within.



The corner nodes will expand the item in two directions (height and width). The four center nodes, one on each side of the item, will expand the item in its respective direction (i.e. top and bottom vertically, left and right horizontally). The way an object is resized can be modified by holding down either the shift or ctrl key.



Holding Shift, while dragging a corner node, will force the height and width to expand proportionally. It has no effect on the four center nodes.



Holding ctrl, while dragging a corner node, will force the height and width to expand proportional in both directions as if the center is pinned.

When dragging the four center nodes it will cause the item to expand in both directions of the current plane (i.e. the top or bottom node will force the field to grow outward, both up and down, vertically and the left or right node will force the field to grow outward, both left and right, horizontally).



**Expand Modifiers**

Category	Modifier
Top Edge	
	Expand Top to Bottom
	Expand Top to Center
	Expand Top to Top
Left Edge	
	Expand Left to Center

	Expand Left to Left
	Expand Left to Right
Bottom Edge	
	Expand Bottom to Center
	Expand Bottom to Top
	Expand Bottom to Bottom
Right Edge	
	Expand Right to Center
	Expand Right to Left
	Expand Right to Right

**Relative Expand**

Relative Expand means that an item's size is defined in **relation** to another item. Therefore when an item is moved all other items whose size is relative to the first will grow or shrink as well.



## 2.10. XFDL Exercise 2 – Alignment

1. Open Workplace Forms Designer 2.6
  - a. Start > Program Files > Workplace Forms Designer 2.6 > Designer
2. Switch to the Resource Perspective and open **XFDL\_Ex02\_fixFormLayout.xfd** from the **TrainingExercises\_XFDL** project in the Navigator View.
3. Use the Outline View to help locate the objects by name, if you select an object in the tree view it will be highlighted in the Design View.
4. Modify LINE1 and LINE2 so they meet in the upper left corner of the form.
  - a. Select LINE1 and then (holding the Shift key) select LINE2, right-click either object and choose **Absolute Align > Top to Top**
  - b. While holding Shift select LINE1 (so that it becomes the reference point – the black dots will surround LINE1), right-click on either object and choose the **Absolute Align > Left to Left**
5. Modify LINE1 and LINE4 so they meet in the lower left corner of the form.
  - a. Select LINE1 and then (holding the Shift key) select LINE4, right-click either object and choose **Absolute Align > Bottom to Bottom**
  - b. While holding Shift select LINE1 (so that it becomes the reference point – the black dots will surround LINE1), right-click on either object and choose the **Absolute Align > Left to Left**
6. Modify LINE2 and LINE3 so they meet in the upper right corner of the form.
  - a. Select LINE3 and then (holding the Shift key) select LINE2, right-click either object and choose **Absolute Align > Top to Top**
  - b. Right-click either object and choose the **Absolute Align > Right to Right**
7. Modify LINE3 and LINE4 so they meet in the lower right corner of the form.
  - a. Select LINE3 and then (holding the Shift key) select LINE4, right-click either object and choose **Absolute Expand > Bottom to Bottom**
  - b. While holding Shift select LINE3 (so that it becomes the reference point – the black dots will surround LINE3), right-click on either object and choose the **Absolute Expand > Right to Right**
8. Modify LINE5 and LINE7 so they span the distance between LINE1 and LINE3.
  - a. Holding the Shift key, select LINE5, LINE7, and LINE1. Right-click and choose **Absolute Align > Left to Left**

- b. Holding the Shift key, select LINE3. Now release the shift key and hold down the ctrl key, select LINE1 to deselect it. Right-click one of the selected objects and choose **Absolute Expand > Right to Right**
9. Modify LINE6 so it spans the distance between LINE2 and LINE7.
  - a. Select LINE6 and LINE2, right-click and choose the **Absolute Align > Top to Top**
  - b. Select LINE7 and deselect LINE2, right-click and choose the **Absolute Expand > Bottom to Bottom**
10. Move LABEL1 so that it is positioned in the top left corner (under LINE2 and after LINE1).
  - a. Select LABEL1 and LINE1, right-click and choose **Absolute Align > Left to Right**
  - b. Select LINE2 and deselect LINE1, right click a selected item and choose **Absolute Align > Top to Bottom**
11. Move/Expand FIELD1 so that it fits below LABEL1 and within the box created by LINE1, LINE2, LINE5, and LINE6.
  - a. Select FIELD1 and LABEL1, right-click and choose the **Absolute Align > Below**
  - b. Select LINE6 and deselect LABEL1, right-click and choose the **Absolute Expand > Right to Left**
12. Move LABEL2 so that it is positioned in the top left corner (under LINE5 and after LINE1).
  - a. Select LABEL2 and LINE1, right-click and choose **Absolute Align > Left to Right**
  - b. Select LINE5 and deselect LINE1, right click a selected item and choose **Absolute Align > Top to Bottom**
13. Move/Expand FIELD2 so that it is exactly the same size as FIELD1, and place it directly below LABEL2
  - a. Select FIELD2 and FIELD1, right-click and choose **Absolute Expand > Make Same Width**
  - b. Select FIELD2 and LABEL2, right-click and choose **Absolute Align > Below**
14. Move LABEL3 so that it is positioned in the top left corner under LINE2 and after LINE6.
  - a. Select LABEL3 and LINE6, right-click and choose **Absolute Align > Left to Right**
  - b. Select LINE2 and deselect LINE6, right click a selected item and choose **Absolute Align > Top to Bottom**
15. Move/Expand FIELD4 so that it fits below LABEL3 and within the box created by LINE5, LINE2, LINE3, and LINE6.
  - a. Select FIELD4 and LABEL3, right-click and choose the **Absolute Align > Below**

- b. Select LINE3 and deselect LABEL3, right-click and choose the **Absolute Expand > Right to Left**
16. Move LABEL4 so that it is positioned in the top left corner (under LINE5 and after LINE6).
  - a. Select LABEL4 and LINE6, right-click and choose **Absolute Align > Left to Right**
  - b. Select LINE5 and deselect LINE6, right click a selected item and choose **Absolute Align > Top to Bottom**
17. Move/Expand FIELD5 so that it is exactly the same size as FIELD4, and place it directly below LABEL4
  - a. Select FIELD5 and FIELD4, right-click and choose **Absolute Expand > Make Same Width**
  - b. Select FIELD5 and LABEL4, right-click and choose **Absolute Align > Below**
18. Align FIELD3 directly below LINE7, and expand it until it touches LINE3 and LINE4.
  - a. Select FIELD3 and FIELD2, right-click and choose the **Absolute Align > Below**
  - b. Select LINE3 and deselect FIELD2, right-click and choose the **Absolute Expand > Right to Left**
  - c. Select LINE4 and deselect LINE3, right-click and choose the **Absolute Expand > Bottom to Top**

## 2.11. Tab Order

Tab order refers to the path that the cursor takes through the input items on a form as the user presses the “Tab” button. By default, the tab order is the order in which the input items were created, which is commonly referred to as the “build order”. When the build order does not properly reflect the tab order required by the business case it becomes necessary for the form designer to change it.



To change the tab order you must first enter into the “tab mode” by pressing the “Show Next Tab Order” button. This button is a toggle that shows the next order, the previous order or hides the tab order.



Once the tab order has been activated a new selection tool appears in the palette. The Connection Creation allows the user to define the next object in the tab order by clicking on first object and then the second object. As items are added to the tab order the dotted lines will be replaced with solid.

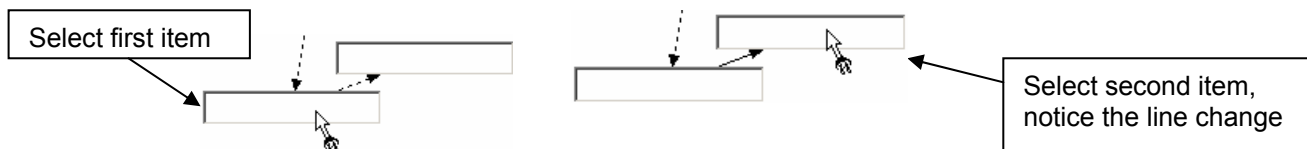
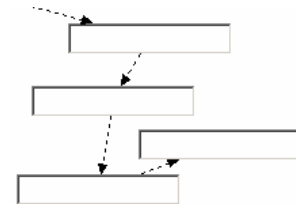


The tab “indicator” lines can also be manipulated directly; from the Palette change the cursor type to **Select** and then click a line to select it. Once a line has been selected edit points will appear at both ends, these edit points can be dragged over other objects on the Design canvas. Moving the lines in this manner will modify the tab order elements.

### Show Next



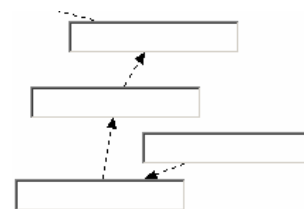
Activating the button shows the default “next” tab order. The dotted lines indicate the direction of the tab order.



### Show Previous



Activating the button a second time shows the default “previous” tab order. The dotted lines indicate the direction of the tab order.



## 2.12. XFDL Exercise 3 - Modifying the Tab Order

When creating an electronic form, an important final check is to determine if the tab order is correct. Everyone has their way of designing a form, some like to design the form with the tab order in mind. Others prefer to place multiple like items on the form and then worry about tab order at the end of the form development process.

Workplace Forms Designer 2.6 has 3 methods for handling tab order, manually moving the tab arrows, using the 'Connection Creation Tool', or modifying the build order.

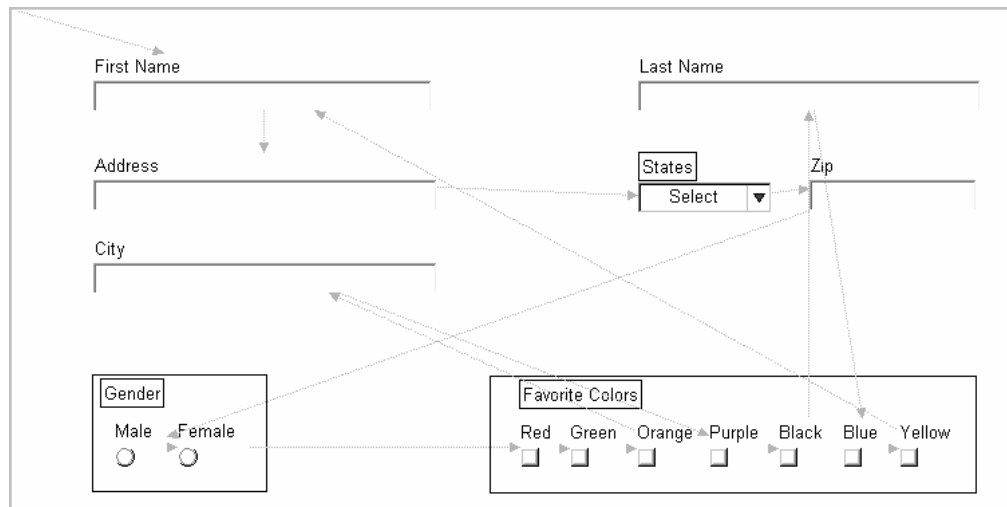
The tab order will go from left to right and from top to bottom for this exercise.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Switch to the Resource Perspective and open **XFDL\_Ex03\_TabOrder\_xfdl.xfd** from the **TrainingExercises\_XFDL** project in the Navigator View.

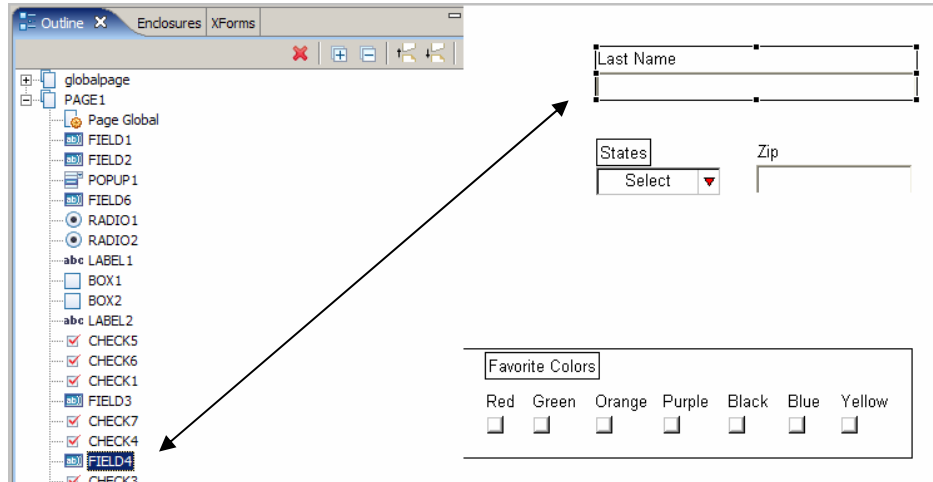
### Build Order Method

In this first method all of the items have been created in order with the exception of two, FIELD4 and FIELD3. Since the default tab order is based on the order depicted in the Outline View we will fix it by rearranging the objects. For small forms this technique is okay, but as you move into more complex forms this is not a very efficient mechanism for setting the tab order.

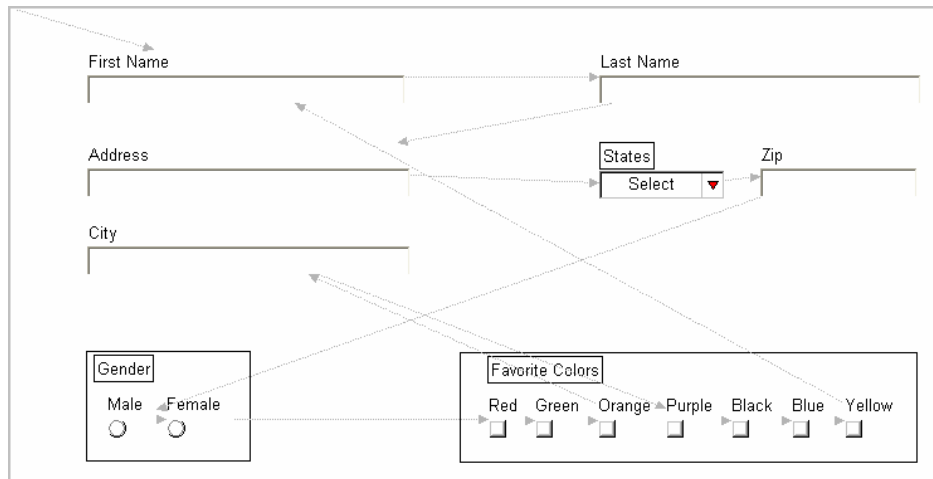
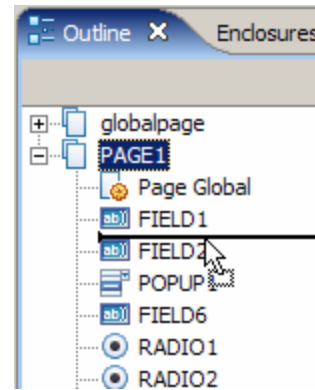
1. To view the tab order, select **View > Show Next Tab Order**
  - a. The designer's work area should now look like the below screenshot when in design mode.



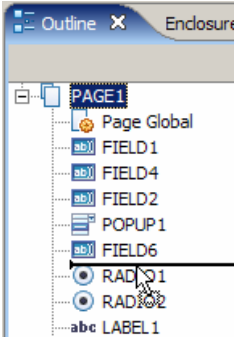
- b. The grey arrows are arranged by the build order. The build order is the order in which the items were placed on the form. You can view the build order via the **Outline View** in the designer.



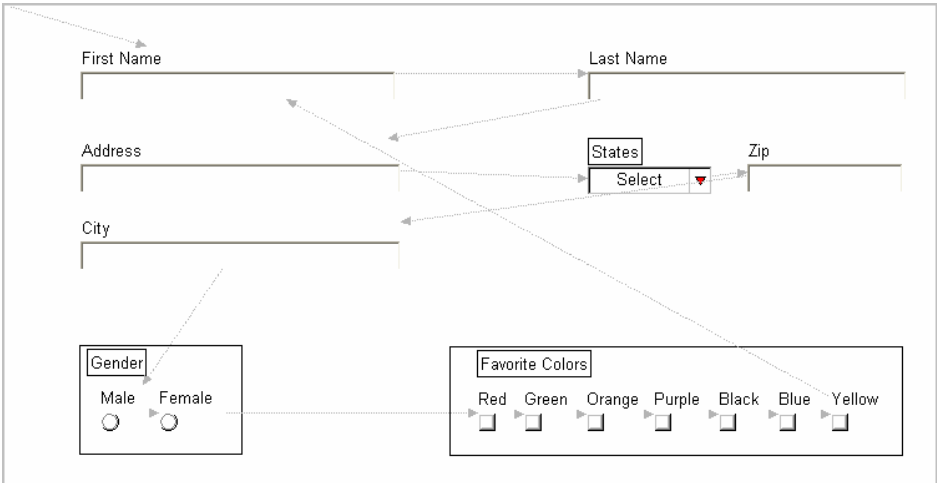
- c. Selecting an item in the design mode will highlight the item in the **Outline View**.
- d. In the form example, FIELD1 is the sid value for First Name and FIELD4 is the sid value of Last Name, to correctly set the tab order, grab the FIELD4 name and drag the item until the black line is shown below FIELD1.
- e. The tab order has now been successfully changed. You can now see the grey arrow points from First Name to Last Name in the screenshot below.



- f. Next, Zip item's tab arrow needs to point to the City item. This is done by moving FIELD3 to below FIELD6 in the **Outline View**.



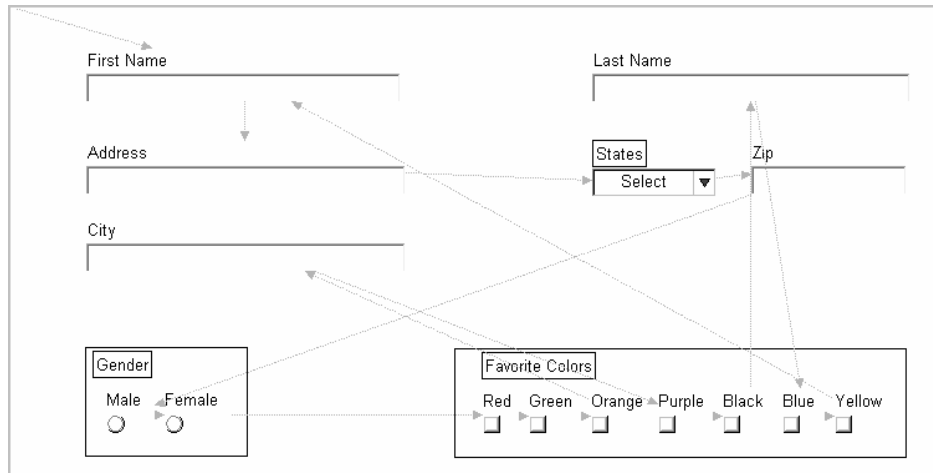
- g. The Tab Order now goes from left to right and from top to bottom (as shown below):





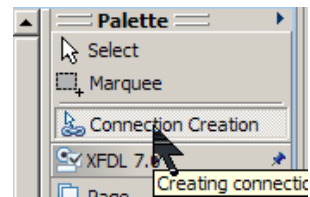
**Connection Creation Tool Method**

Do the exercise again, but this time focus on using the connection creation tool to modify the tab order. This is probably the most efficient method of modifying the tab order of forms with a large number of items.

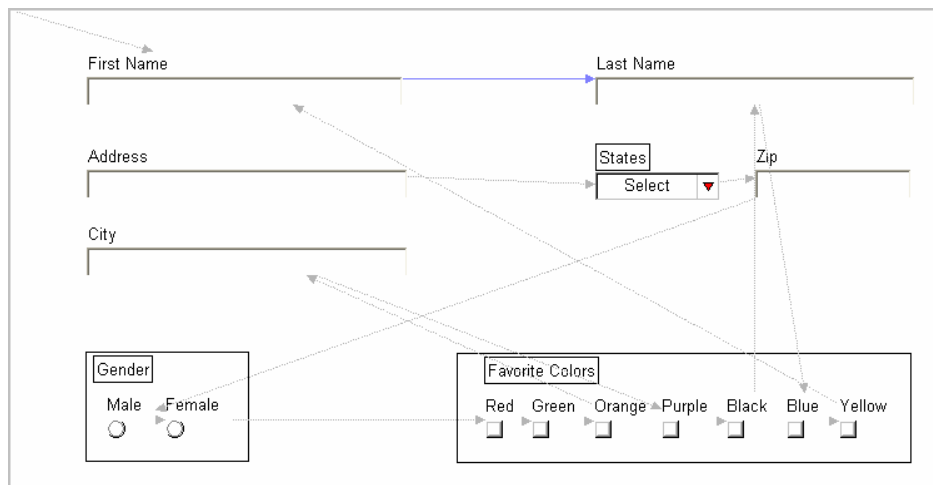
1. Switch to the Resource Perspective and open **XFDL\_Ex03\_TabOrder\_xfdl.xfd** from the **TrainingExercises\_XFDL** project in the Navigator View.
2. To view the tab order, select **View > Show Next Tab Order**
  - a. Designer's work area should now look like the below screenshot when in design mode.



- b. To turn on the Connection Creation Tool, you must activate the tool in the palette.
- c. Once activated the cursor will either look like  when over an item or  when within the workspace but not over an item.

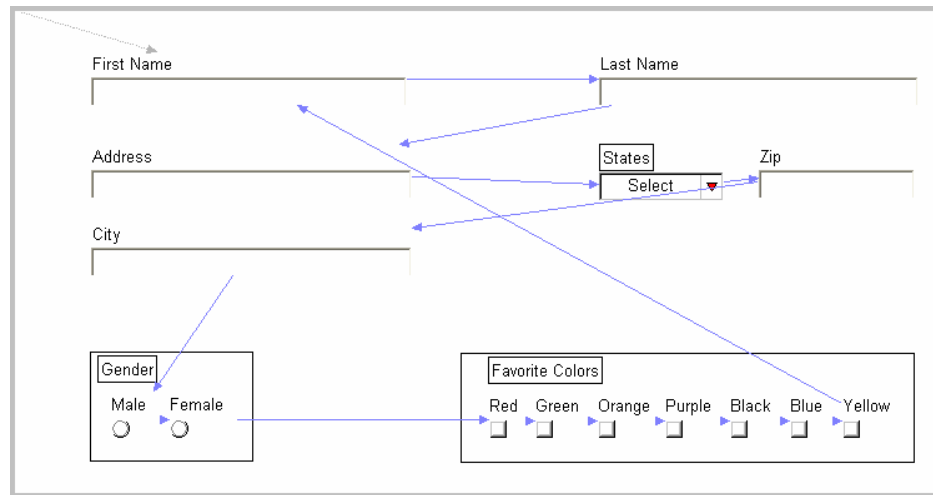


- d. Click the **First Name** item. Note: The action of moving the cursor to the next item will cause the designer to draw a guide line from the original clicked item to the cursor.
- e. Click the **Last Name** item. Note: moving the cursor over an item will cause the guide line to snap to that item.
- f. Last Name item now has a blue arrow pointing to it from First Name item. In code view, the tag `<previous>FIELD1</previous>` has been added to Last Name item.





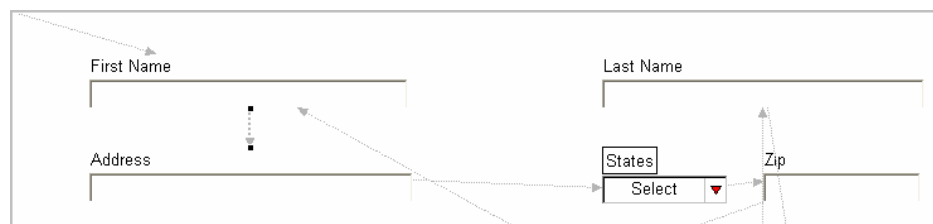
- g. You may now continue the above process to define your tab order. The end result should look similar to the screenshot below.





### Manually modify Tab Order

Do the exercise again, but this time we will focus on modifying the tab order manually.

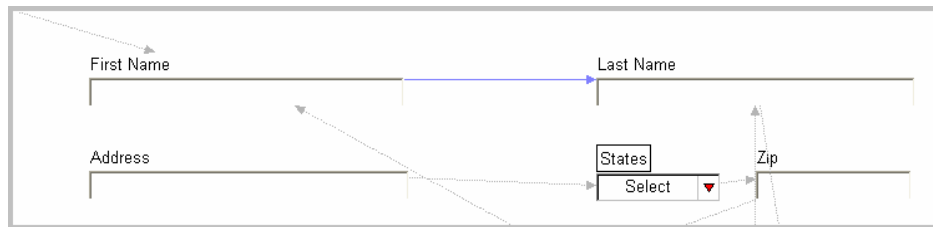
1. Switch to the Resource Perspective and open **XFDL\_Ex03\_TabOrder\_xfdl.xfd** from the **TrainingExercises\_XFDL** project in the Navigator View.
2. To view the tab order, select **View > Show Next Tab Order**
3. To manually modify the tab order:
  - a. Click arrow to Highlight.



- b. Position the cursor at the arrow head of the pointer until the cursor changes from  to 

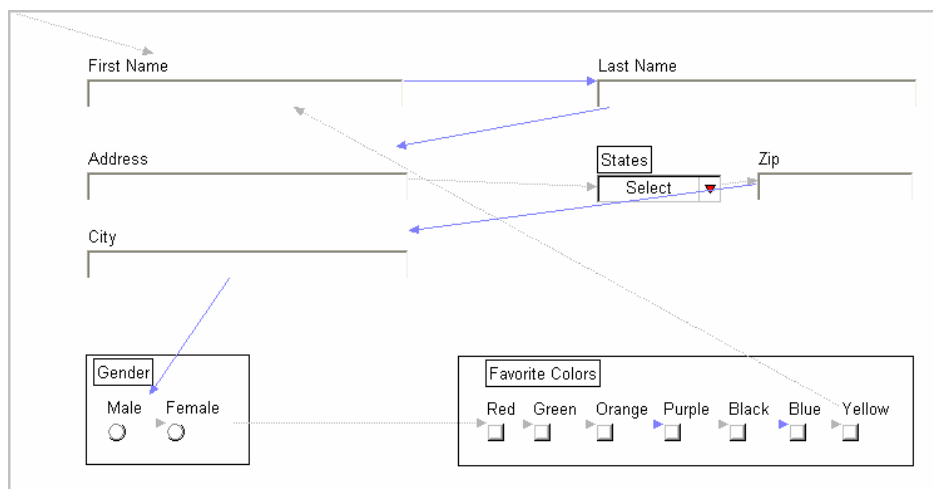


- c. Hold and drag the arrow to the desired item. This will create a blue arrow.



- d. Continue modifying the pointers until you can tab through the form, left to right and from top to bottom.

- e. The forms tab order should look similar to the screenshot below:



## 2.13. Working with Enclosures

The Enclosures View provides the user the ability to embed different objects inside the form. File types that can be imported are images (i.e. gif, png, bmp, tif), jar files, schema files, wsdl files or XForms instances.

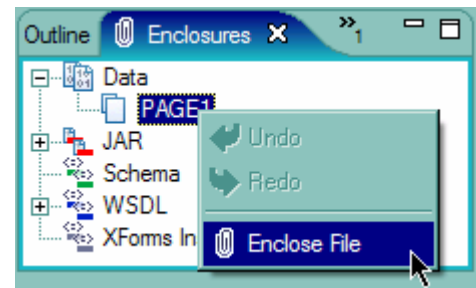
If the Enclosures View is not active then it can be opened by selecting **Window > Show View > Enclosures**. The Enclosures View lists five different categories: Data, JAR, Schema, WSDL, and XForms Instance.

Files that have been added through the Enclosures View can be saved locally or removed from the form by right-clicking the item and selecting the appropriate action.

### 2.13.1. Data

This function provides the ability to embed images within your form.

Within the **Enclosures View**, expand **Data** right-click on the desired page and select **Enclose File**. This will launch a standard file dialog where you can navigate your local file system. Locate the image through this dialog and click **Open**. The file will then appear as a child under the selected page.

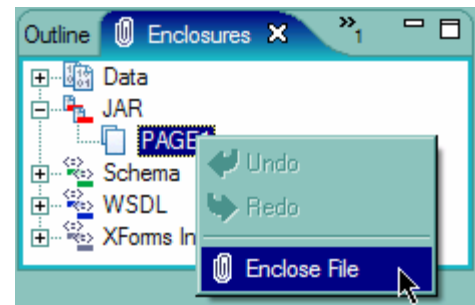


Once an image has been enclosed it can then be applied to a label or a button within your form. If you select the image and drag it onto the canvas then the Designer will create a label that displays the image. You can also drag the image, from the Enclosures View, onto an existing label or button.

### 2.13.2. JAR

JAR is an abbreviation for **Java Archive**; JAR files may contain custom extensions that provide additional functionality. This function provides the ability to embed a JAR file within your form. JAR files contain functions that can be called within XFDL to enhance a form's behavior; usually the functions provide functionality that does not exist within the Viewer.

Within the **Enclosures View**, expand **JAR** right-click on the desired page and select **Enclose File**. This will launch a standard file dialog where you can navigate your local file system. Locate the JAR file through this dialog and click **Open**. The file will then appear as a child under the selected page.



All functions will exist within a package. The package name is defined by the developer who created the extension. If you know the package name and the functions included in the JAR file then you can access them in XFDL (i.e. <packageName>.<function>).

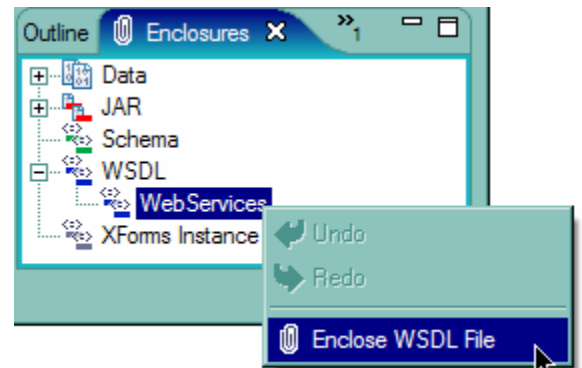
### 2.13.3. Schema

This function provides the ability to embed an XML schema within your form. Schemas can only be enclosed if the form already has an XForms model. Working with XForms model and schemas is not a topic for this module.

### 2.13.4. WSDL

WSDL is an abbreviation for **Web Service Description Language**; it is an xml file that describes a web service and its functions. If you want a form to call a web service then you must first enclose the WSDL that defines it. Once the WSDL has been enclosed you may then access its functions from within XFDL.

Within the **Enclosures View**, expand **WSDL** right-click on **Web Services** and select **Enclose WSDL File**. This will launch a standard file dialog where you can navigate your local file system. Locate the WSDL file through this dialog and click **Open**. The file will then appear as a child under Web Services.



All the Web Service functions will be placed into a package. The package name is a combination of the service name and port name, separated by an underscore (\_). The following XML fragment was taken directly from a WSDL:

```
<service name="AirportWeather">
  <documentation>AirportWeather</documentation>
  <port binding="tns:StationBinding" name="Station">
    <soap:address location=" ... "></soap:address>
  </port>
</service>
```

Given the above fragment the package name for this web service is "AirportWeather\_Station". Once I have the package name I can now access any of the functions that are included in the web service. The references follow the same rules as XFDL functions (i.e. <packageName>.<function>)

### 2.13.5. XForms Instance

This function provides the ability to embed an arbitrary block of XML code as an XForms Instance. XML fragments can only be enclosed if the form already has an XForms model. Working with XForms and instance is not a topic for this module.

## 3. Understanding XFDL

### 3.1. Audience

This section is intended to teach the more advanced syntactical fundamentals of the XFDL language. This module will help the reader to understand topics such as: the XFDL Node Structure, Namespaces, References, Operators, Functions, creating custom options, understanding XFDL events, and writing XFDL computes. Those who are interested in understanding XFDL in greater detail and leveraging that understanding to build dynamic forms should review this section.

### 3.2. Overview

Workplace Form documents are written in the Extensible Forms Description Language (XFDL). XFDL is a language that is built on the XML foundation, but adds an event-based compute engine, the ability to embed attachments and a mechanism for defining the presentation layer. There are several aspects to the XFDL language that are essential to understanding how to design and develop dynamic forms.

### 3.3. Namespaces

The W3C (<http://www.w3.org>) states that "XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references" (<http://www.w3.org/TR/REC-xml-names/>).

What this means is that by using namespaces, we can control the scope of element names within the XML document. In essence, we can have different XML syntaxes within one document.

A namespace declaration consists generally of a prefix followed by a URI. In XFDL documents, the namespaces are defined as attributes to the XFDL declaration, as shown below:

```
<XFDL xmlns="http://www.ibm.com/xmlns/prod/XFDL/7.0"
xmlns:xfdl="http://www.ibm.com/xmlns/prod/XFDL/7.0"
xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom"
xmlns:designer="http://www.ibm.com/xmlns/prod/workplace/forms/designer/2.6">
```

Four namespaces are declared here. The first does not define a prefix and is therefore the default namespace for the XFDL element and all of its children. The URI used is that of the XFDL namespace. This means that all elements and attributes found in this document implicitly belong to the XFDL namespace. Elements will not inherit this namespace if their name is lead by a prefix associated to another namespace or there is another default namespace declaration in the form.

The second namespace declaration defines a prefix for the default (XFDL) namespace. This prefix can be used to explicitly assign an element or attribute to the XFDL namespace.

The third namespace declaration defines a prefix for the custom namespace. This namespace is typically used when adding custom (non-XFDL) options to the form.

The fourth namespace declaration defines a prefix for the Designer's namespace. This namespace is used by the Designer when adding non-XFDL options to the form.

The default namespace is shown by not having a prefix, whereas the last two namespace declarations consist of `xmlns:THEPREFIX="TheURI"`, where the prefix is a short word with no spaces. The URI is commonly a company URL, so that the URI is unique, and information concerning the namespace is often at the URL.

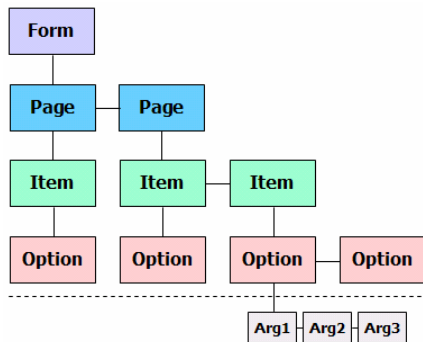
In the example below, 'value' is part of the core XFDL namespace, and therefore does not require the namespace prefix, whereas 'myOption' is not, and therefore the designer must indicate to which namespace it belongs:

```
<value></value>  
<custom:myoption>3</custom:myoption>
```

For more information, see the XFDL specification, or the W3C namespace specification (<http://www.w3.org/TR/REC-xml-names/>).

### 3.4. Node Structure

An XFDL form contains a variety of elements, such as pages, fields, labels, and images. When loaded, the Viewer organizes these elements into nodes. These nodes link to create a top-down hierarchy that is similar in appearance to a family tree. At the top there is a single node that is the parent of all other nodes. As you add new nodes, or children, to the hierarchy, it grows downward and outward. The result resembles an inverted tree, as illustrated in the following diagram:



As you can see, the hierarchy creates natural levels that reflect the types of elements discussed earlier. Nodes of the same type always reside on the same level of the tree. Each node corresponds to the form element of the same name. The levels of this hierarchy, in descending order, are:

**Form** - This is the highest level of the tree. It always consists of just one node: the form node, which is created automatically. All other nodes are descendants of it.

**Page** - This level always contains a minimum of two page nodes. The first is a special page node called the global page node. This node specifies configuration settings for the form as a whole. The second node represents the first page in the form. As you create new pages in your form, each additional page is represented by a new page node.

**Item** - Item nodes are descendants of page nodes. Items are always contained within a page. A page can have an unlimited number of item nodes.

**Option** - Option nodes are the children of item nodes, and are always contained within their parent item or page node. You can also place custom options within page or item nodes. An item can have an unlimited number of option nodes.

**Argument** - These nodes store settings for option nodes, and are always contained by their parent option node. Argument nodes can contain other argument nodes. There can be an unlimited number of sub-levels in the argument level.

### 3.5. References

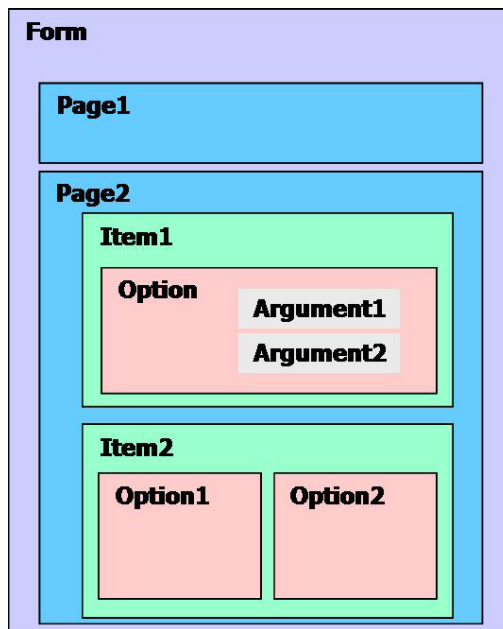
Learning to write XFDL references is fundamental to understanding how to build dynamic forms. Properly formed references allow you to refer to specific values or content in a form. This means that you can access data from any item in the form, which can then be used to determine the content of another option. You may use references to copy existing data or use it in a calculation.

A reference, like the node structure, has four levels: page, item, option and argument. It may help to think of each level as containers, where each container is held within its parent. The key to creating your reference is to “open” all the containers to get to the one you are looking for. When constructing your reference the first three levels, the SID of each container is appended together and separated by a period. The syntax changes slightly once we descend to the option container because it is an array of arguments. Arguments, within the reference, are denoted by using square brackets (i.e. PAGE.ITEM.OPTION[ARGUMENT]).

XFDL supports an infinite level of arrays. An example of an option that contains an array of arguments is the itemlocation:

```
<itemlocation>
  <x>8</x>
  <y>133</y>
  <width>217</width>
  <height>38</height>
</itemlocation>
```

In XFDL, array arguments can be accessed by the name or index of the argument. Array indexes start with zero. Therefore the reference that retrieves the value of “y” is “itemlocation[y]” or “itemlocation[1]”





### 3.6. XFDL Exercise 4 – Writing XFDL References

At this point you should understand the basics of how to build simple references. A worksheet has been included in this training package that will help you to apply your new found knowledge. Please take a few minutes to work through the questions.

1. Open the exercise form, **XFDL\_Ex04\_XFDLReferences.xfdl**, in the Viewer.
2. Answer each question by typing your reference in each field
  - a. If your answer is correct it will be indicated by a green label.
  - b. The solutions can be seen by pressing the buttons corresponding with each question.

## 3.7. Operators

The following tables include all the valid operators that can be used within XFDL.

### 3.7.1. Numeric

Operand	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (returns the remainder)
^	Exponential

### 3.7.2. Logical

Operand	Description
or	Boolean Or
and	Boolean And
not	Boolean Not

### 3.7.3. Miscellaneous

Operand	Description
+.	String Concatenation
+ or +.	Function Concatenation

### 3.7.4. Relational

Operand	Description
==	Equal To
!=	Not Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

## 3.8. Computes

Computes are powerful tools that add intelligence to XFDL forms, by providing built-in logic and the ability to perform calculations. The ability to include mathematical and conditional computes gives form designers a great deal of flexibility in the form's user interface. Using computes, you can make dynamic e-forms that present different interfaces based on the users' unique needs.

All computes rely heavily on node referencing, which allows form elements to 'copy', 'share', or 'update' information from other form elements. You can add computes to a form by using the Designer's Properties View or directly to the XFDL code in the Source View.

### 3.8.1. Mathematical Computes

Typically, form designers want to reduce the potential for user errors. You can drastically reduce the potential for mathematical errors by allowing the form to perform all required mathematical computes. For example, you could create a tax form that automatically performed all of the necessary calculations. As a result, users would only need to enter the correct starting values to complete the form.

You create mathematical formulas by embedding computes in the form. As the user fills in the form, the computes run automatically. Mathematical computes include addition, subtraction, string concatenation, multiplication, division, integer modulus, sum, unary minus, and exponentiation.

### 3.8.2. Conditional Computes

The second type of XFDL compute is the Conditional Expression, or Decision Statement. Like mathematical computes, these conditional computes are embedded in the XFDL code and their execution occurs on the user's computer.

Conditional computes are often referred to as IF – THEN – ELSE expressions. They are symbolically written as "x ? y : z"; where x is a boolean expression and y is what occurs when x evaluates true and z when false. . You can read this as:

```
If x THEN y ELSE z
```

In other words: IF condition x is true THEN use condition y. Otherwise use condition z. For example, if you wanted a field on page 2 to copy information from a field on page 1 if a checkbox is selected, you would use the following compute to the field on page 2:

You could read the above compute as:

If the checkbox is turned on, then place the value of PAGE1.FIELD1 into the value of PAGE2.FIELD2. Otherwise, leave PAGE2.FIELD2 blank.

```
<value compute="PAGE1.CHECK1.value == 'on'  
? PAGE1.FIELD1.value  
: ''" />
```

Empty single quotes indicate an empty value. In other words, if CHECK1 is not selected, the compute does nothing.

### 3.8.3. Creating Computes in the Designer

The Designer contains a compute wizard that provides the user the ability to create several different types of computes; these computes have been broken into six categories:

**The value is equal to another form item** - Allows you to set the value of an option equal to a property of another item on the form.

**The value is equal to a function** - Allows you to create a compute that: Assigns the value returned by a function such as set.

**Set by a calculation of 2 values** - Allows you to create a compute that calculates the value of the option from two other values using any operator (add, subtract, divide, exponent, etc.). The two values can come from the properties of other items, numbers you provide, the results of function calls, or combinations of these.

**The value is equal to the sum of multiple fields on a form** - Allows you to create a compute that sets the option to equal the total value of selected fields.

**Determined by a decision (If/Then/Else)** - Allows you to create a compute that will evaluate a condition or test (for instance, is FIELD2.active equal to “on”) and assign the option a certain value if the condition is “true” and a different value if the condition is “false”. The condition and the values used in the compute can be based on items’ values, numbers or text you provide, or the results of function calls.

**Set by a manually created formula** - Allows you to type out the formula manually. This method allows more flexibility, and the capability to combine more than one type of formula, but it means that you must be familiar with XFDL syntax.

Once you have selected the type of formula you want to create, the Formula dialog box automatically populates with text, fields, and choices that correctly structure your formula and assist you in selecting the proper properties, functions, and values for your compute.

### 3.9. XFDL Exercise 5 – Writing Computes

This lesson is meant to introduce you to the designer's compute wizard and show you how to create a simple XFDL compute. Computes are powerful tools that add intelligence to XFDL forms by providing built-in logic and the ability to perform calculations. The capability to include mathematical and conditional computes gives form designers a great deal of flexibility in the form's user interface. This exercise will expose you to five of the most common types of XFDL computes.

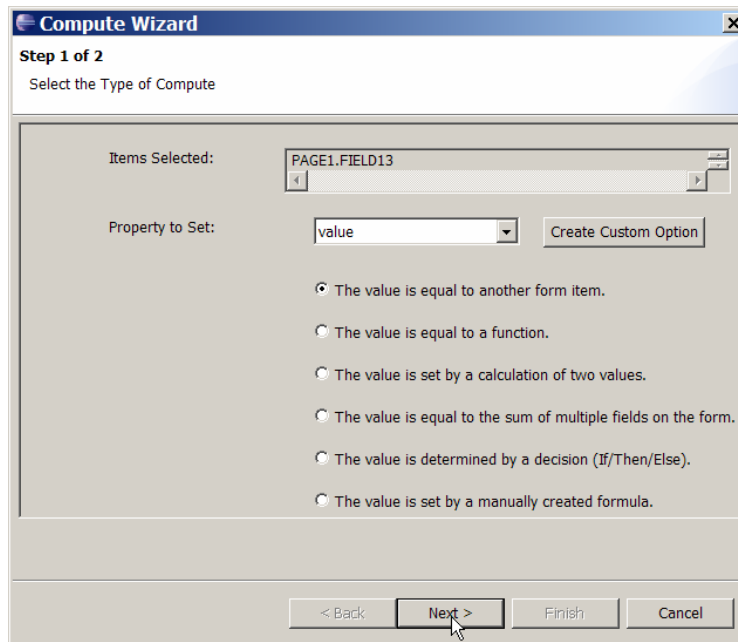
This document has been written assuming little to no previous knowledge of XFDL. This tutorial assumes that the reader has some basic understanding of the Eclipse interface.


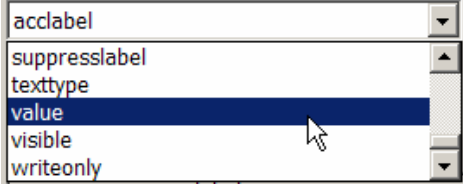
1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
  - b. Switch to the Resource Perspective and open **XFDL\_Ex05\_XFDLComputes.xfd** from the **TrainingExercises\_XFDL** project in the Navigator View.

#### First Compute: Setting a value equal to the value of another item.

Equal to the value of an item	
Name	Copy of name
<input type="text"/>	<input type="text"/>

1. Right click FIELD2 (under **Copy of name**) and select **Wizards > Compute Wizard** to open the following window:





2. Select **value** from the dropdown list and select the first radio button “**The value is equal to another form item.**”
3. Press the **Next >** button then select the  button. The designer should switch back to the design view where you should select FIELD1 (under **Name**).
4. Once you select the field the application should bring you back to the wizard page where you need to once again select the **value** option from the drop down menu.
 
5. Click the **Finish** button to complete your first compute.


### Second Compute: Setting a value equal to a function.

Equal to a function

Name	String length
<input type="text"/>	<input type="text"/>

1. Right click FIELD4 (under **String Length**) and select **Wizards > Compute Wizard**
2. Select **value** from the dropdown list and select the second radio button “**The value is equal to a function.**” Then click **Next >**.
3. Click the **Function** button
4. Select **strlen** from the function call popup. 
5. Select “**Choose an item on the form.**” Then select the  button. The designer should switch back to the design view where you should click on FIELD3 (under **Name**).

string:

Choose an item on the form. ▾		value ▾	PAGE1.FIELD3.value
-------------------------------	-------------------------------------------------------------------------------------	---------	--------------------



6. Click **OK** then click the **Finish** to complete your second compute.

### Third Compute: Set by a calculation of two values.

Set by a calculation of two values


Multiplicand	Multiplier	Product
<input type="text"/>	<input type="text"/>	<input type="text"/>

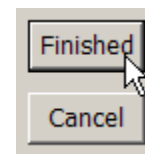
1. Right click FIELD13 (under **Product**) and select **Wizards > Compute Wizard**

2. Select **value** from the dropdown list and select the third radio button “**The value is set by a calculation of two values.**” Then click **Next >**.
3. For the **First Value:** select “**Choose an item on the form.**” Then select the  button. The designer should switch back to the design view where you should click on FIELD5 (under **Multiplicand**).
4. Select **value** from the drop down list
5. For the Function choose “\* (multiplied by)”.
6. For the **Second Value:** select “**Choose an item on the form.**” Then select the  button. The designer should switch back to the design view where you should click on. FIELD6 (under **Multiplier**).
7. Select **value** from the drop down list.
8. Click the **Finish** button to complete your third compute.

#### Fourth Compute: Equal to the sum of multiple fields on the form.

Equal to the sum of multiple fields			
A	B	C	Total
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>


1. Right click FIELD10 (under **Total**) and select **Wizards > Compute Wizard**
2. Select **value** from the dropdown list and select the third radio button “**The value is equal to the sum of multiple fields on the form.**” Then click **Next >**.
3. Select the  button. The designer should switch back to the design view. Hold down the ctrl key and select FIELD7, FIELD8, and FIELD9. Then click **Finished** in the box at the top left of your screen.



PAGE1.FIELD10.value =

PAGE1.FIELD7  
 PAGE1.FIELD8  
 PAGE1.FIELD9

}

  
 value  
 Delete

Source Code Being Generated:

```
PAGE1.FIELD7.value + PAGE1.FIELD8.value + PAGE1.FIELD9.value
```



4. Click the **Finish** button to complete your fourth compute.





### Fifth Compute: Determined by an if statement


Determined by an if statement

A	B	Maximum value
<input type="text"/>	<input type="text"/>	<input type="text"/>


1. Right click FIELD14 (under **Maximum value**) and select **Wizards > Compute Wizard**
2. Select **value** from the dropdown list and select the third radio button “**The value is determined by a decision (if/then/else).**” Then click **Next >**.
3. For the **If** section in the first popup select “**Choose an item on the form.**” Then select the  button. When you return to the form click on FIELD11 (under **A**). Then Select **value** from the next popup.
4. Select “**Choose an item on the form.**” Then select the  button. When you return to the form click on FIELD12 (under **B**). Then Select **value** from the popup
5. In the middle popup on the bottom line select **> (greater than)**


If

Choose an item on the form.		value	Choose an item on the form.		value
PAGE1.FIELD11.value	> (greater than)	PAGE1.FIELD12.value			


6. For the **Then** section in the first popup select “**Choose an item on the form.**” Then select the  button. When you return to the form click on FIELD11 (under **A**). Then Select **value** from the popup.

Then

Choose an item on the form.		value
PAGE1.FIELD14.value =	PAGE1.FIELD11.value	

7. For the **Else** section in the first popup select “**Choose an item on the form.**” Then select the  button. When you return to the form click on FIELD12 (under **B**). Then Select **value** from the popup.

Else

Choose an item on the form.		value
PAGE1.FIELD14.value =	PAGE1.FIELD12.value	

8. Click the **Finish** button to complete your fifth compute.

## 3.10. Functions

Functions allow forms to perform procedural logic and complex operations that would normally require complex conditional statements to achieve. For example, you can use functions to count the number of lines a string would take up over a given width, return a substring of a string, or set the value of a form option.

You can choose to use a predefined system function or you can create your own. User-defined functions are discussed in the API training modules.

All functions are grouped into packages. The Viewer includes two default packages called “viewer” and “system”. Both packages contain functions that can be used within XFDL. They include standard mathematical operations, date and time functions, some generic utility functions and functions that manipulate a string. You can call them from any compute in your form and you can add them using the Designer’s Function dialog box or hand code them in Code View or other text editor.

Parameters are used to pass information to functions. For example, parameters can define the option or string that the function will evaluate and the function’s limits, such as its start and end point. These parameters are placed in parentheses immediately following the function name. Each parameter is contained in quotation marks and separated by a comma:

```
function('param_1', 'param_n')
```

Refer to the XFDL Specification for additional detail on implementing functions within XFDL.

### 3.10.1. Explore XFDL Functions

Take a moment and open the XFDL specification. Have a look at the section labeled “Details on Function Calls”; all the functions that are available within the XFDL language have been categorized and explained in detail. You will find that there are functions for manipulating strings, working with dates and times, calculating mathematical values and some that couldn’t be categorized (so were thrown together in a miscellaneous category). If you have a good idea of the type of functions that are available then you will be much better equipped to design your dynamic forms!

If you do not currently have the Workplace Forms XFDL Specification it can be downloaded from <http://www.ibm.com/software/lotus/support/workplaceforms>. Click on the **Product Documentation** link under the **Primary support resources** heading.

## 3.11. XFDL Example 6 – Using XFDL Functions

This lesson is meant to introduce you to the functions that are available within the XFDL language. This exercise will expose you to five groupings of functions that are available and some of the most commonly used.

This document has been written assuming little to no previous knowledge of XFDL. This tutorial assumes that the reader has some basic understanding of the Eclipse interface.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Switch to the Resource Perspective and open **XFDL\_Ex06\_XFDLFunctions.xfdl** from the **TrainingExercises\_XFDL** project in the Navigator View.

### Function 1 – viewer.messagebox

This function displays a message box that prompts the user. In this exercise we will call this function, when the user changes the value in the **Change Value** field.

#### Method Call

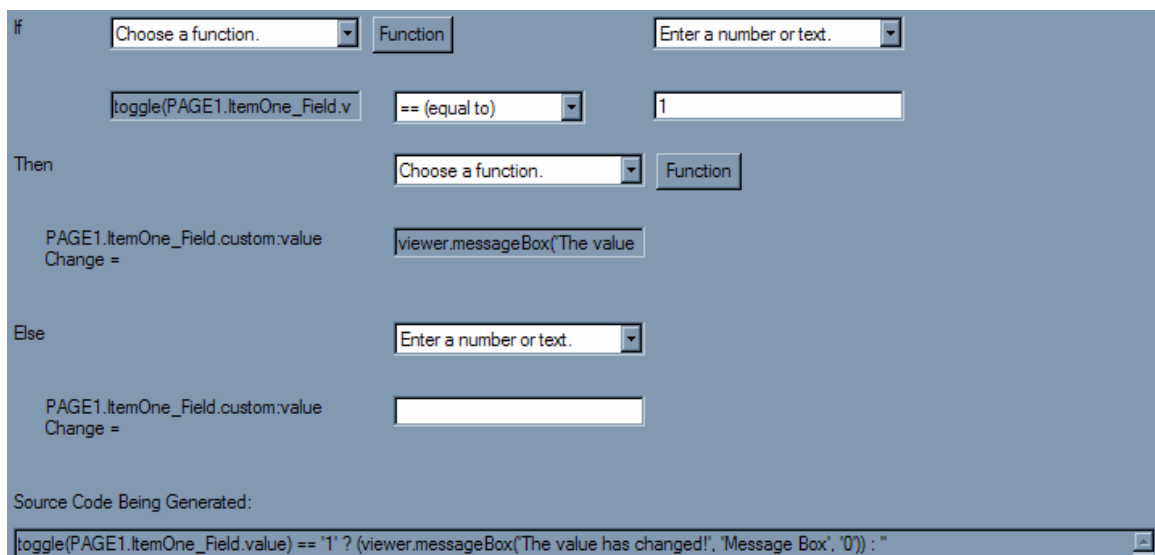
**messageBox**(*message*[, *caption*, *messagetype*])

Where *message* **string** contains the message to display in the main portion of the message box. *Caption* **string** contains the caption to display in the title bar of the message box, and the *messagetype* specifies whether the message box is an **OK** or a **QUESTION** box. If the type is **OK**, then the box will contain an OK button. If the type is **QUESTION**, then the box will contain a Yes button and a No button. The default type is OK.

1. Right click Change Value field (under the viewer.messageBox label) and select **Wizards > Compute Wizard**.
2. Create a new custom option called 'valueChange'.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
4. For the **If** statement:
  - a. In the first drop down, select '**Choose a function.**'
  - b. Click the **Function** button. A new window is opened 'Source Code Being Generated'.
  - c. In the drop down, select **Toggle**.
  - d. In the Parameters section under references, select "**Choose an item on the form**",
  - e. Select the hand icon

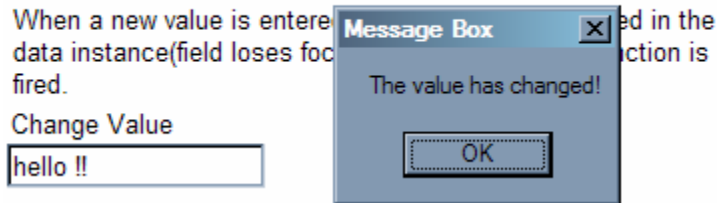
- f. Click the **Change Value** field
  - g. Click **Finish**.
  - h. Select **Value** in the drop down next to the hand icon.
  - i. Select **OK**.
  - j. Select “**Enter a number or text**” in the 2<sup>nd</sup> If drop down and enter the value of **1**.
  - k. At this point, the Source Code Being Generated should look like this:
 

```
toggle(PAGE1.ItemOne_Field.value) == '1' ? ('') : ''
```
5. In the Then drop down box, select ‘**Choose a function.**’ And select the **Function** button.
    - a. A new window has been created. In the drop down box, select the function ‘**viewer.messageBox**’.
    - b. In the message parameter drop down, select ‘**Enter a number or text**’ and enter the value **The value has changed!**
    - c. In the caption parameter drop down, select “**Enter a number or text**” and enter the value **Message Box**.
    - d. In the messagetype parameter drop down, select “**Enter a number or text**” and enter the value **0**.
    - e. Select **OK**.
  6. In the Else drop down box, select ‘**Enter a number or text.**’ And leave the value empty.
  7. Your compute wizard should look like the screen shot below.



8. Select **Finish**.
9. View the form in preview tab to test it. Enter text in the **Change Value** field and tab out of the field to view the viewer.messageBox action.

## viewer.messageBox



### Function 2 – isValidFormat

This function returns the boolean result of whether a string is valid according to the setting of the format option referred to in *formatOptionReference*.

This exercise uses a button to fire the function isValidFormat to check the format of the Current Date field. "0" is displayed if the format is invalid and "1" otherwise.

### Method Call

**isValidFormat**(string, formatOptionReference)

Where **string** is a string to be checked against the format, and **formatOptionReference** is the option reference of the format, including the page sid if necessary, to check the string against.

1. Right click **Validate button** (under the isValidFormat label) and select **Wizards > Compute Wizard**.
2. Create a new custom option called 'onClick'.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
4. For the If statement:
  - a. In the first drop down, select '**Choose a function.**'
  - b. Click the **Function** button. A new window is opened 'Source Code Being Generated'.
  - c. In the drop down, select **Toggle**.
  - d. In the Parameters section under references, select "**Choose an item on the form**"
  - e. Select the hand icon

- f. Click the **Validate** button
  - g. Click **Finish**.
  - h. Select **activated** in the drop down next to the hand icon.
  - i. In the **Start** section, choose "**Enter a number or text**" and enter "off" in the text box.
  - j. In the **End** section, choose "**Enter a number or text**" and enter "on" in the text box.
  - k. Select **OK**.
  - l. Select "**Enter a number or text**" in the 2<sup>nd</sup> If drop down and enter the value of **1**.
5. In the Then drop down box, select '**Choose a function.**' And select the **Function** button.
- a. A new window has been created. In the drop down box, select the function '**isValidFormat**'.
  - b. In the string parameter drop down, select "**Choose an item on the form**", Select the hand icon then click on the **Current Date** field.
  - c. Make sure the text field next to the hand tool says **Value**.
  - d. In the formatOptionReference parameter drop down, select "**Choose an item on the form**", Select the hand icon, Click **Current Date** field
  - e. Click **Finish**.
  - f. Make sure the text field next to the hand tool says **Format**.
  - g. Click **OK**.

6. In the Else drop down box, select 'Enter a number or text.' And leave the value empty.
7. Your compute wizard should look like the screen shot below.

8. Click **Finish**.
9. Now click on the **Source** tab and add a viewer.messageBox function to the **custom:onClick** option as follows:

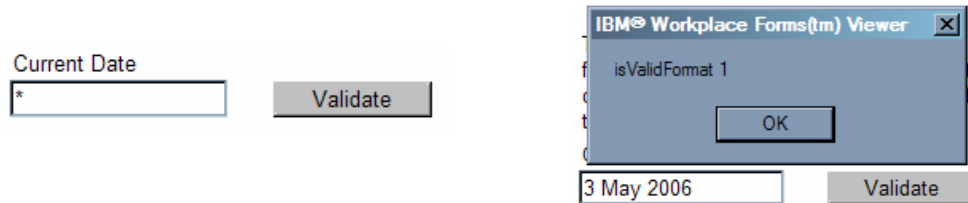
From:

```
<custom:onClick xfdl:compute="toggle (PAGE1.setIndex1.activated,
'off', 'on') == '1' ? (isValidFormat (PAGE1.FIELD3.value,
'PAGE1.FIELD3.format')) : ''"></custom:onClick>
```

To:

```
<custom:onClick xfdl:compute="toggle (PAGE1.setIndex1.activated,
'off', 'on') == '1' ? viewer.messageBox ('isValidFormat ' +
isValidFormat (PAGE1.FIELD3.value, 'PAGE1.FIELD3.format')) :
''"></custom:onClick>
```

10. To test the form click on the **preview** tab. Enter \* in the **current date** field to fill it with the current date and click the **validate** button.



### Function 3 – set

This function sets the value of an XFDL form option or of an element in the XForms model. This exercise uses the set function to push the Current Date under isValidFormat to the field “Push Current Date here”.

### Method Call

**set**(reference, value, referenceType, scheme)

The first parameter, **reference**, is the string reference of the element being set, and **value** is the reference’s new value. The **referenceType** and **scheme** are optional and can be left out for this example.

1. Right click **Push value button** (under the set label) and select **Wizards > Compute Wizard**.
2. Create a new custom option called ‘onClick’.
3. Select the radio button ‘The value is determined by a decision (If/Then/Else)’ and then click **Next >**.
4. For the If statement:
  - a. In the first drop down, select ‘**Choose a function**’ and click **Next >**.
  - b. Click the **Function** button. In the new dialog select **toggle** from the function call dialog.
  - c. In the Parameters section under references, select “**Choose an item on the form**”, Select the hand icon, Click **Push Value** button. If the dialog does not come back into focus then press “Finished”.
  - d. Select **activated** in the drop down next to the hand icon.
  - e. Set the **Start** parameter to “**Enter a number or text**” and enter “off” in the text box.
  - f. In the **End** section, choose “**Enter a number or text**” and enter “on” in the text box.
  - g. Select **OK**.
  - h. Select “**Enter a number or text**’ in the 2<sup>nd</sup> If drop down and enter the value of **1**.



5. In the Then drop down box, select '**Choose a function.**' And select the **Function** button.
  - a. A new window has been created. In the drop down box, select the function '**set**'.
  - b. In the **reference** parameter drop down, select "**Choose an item on the form**", Select the hand icon, Click **Push Current Date Here** field.
  - c. In the **value** drop down, select "**Choose an item on the form**", Select the hand icon, Click the **Current Date** field
  - d. Click **Finish**.
  - e. Click **OK** to close the window. The compute must look like this at the end of this step.

```
toggle(PAGE1.PushValue_Button.activated, 'off', 'on') == '1' ? (set(PAGE1.Other_Field.value, PAGE1.FIELD3.value)) : "
```

6. Click **Finish** to exit the Compute Wizard.
7. Test the form in the **preview** tab. Enter \* in the **Current Date** field under **isValidFormat** label to fill it with the current date and click on **Push Value** button under the **set** label.

Current Date

**set**

This exercise uses the set function to push the Current Date under isValidFormat to the field below.

Push Current Date here.

#### Function 4 – Power

This exercise uses the power function to return the number represented in *number* raised to the power indicated by *power*.

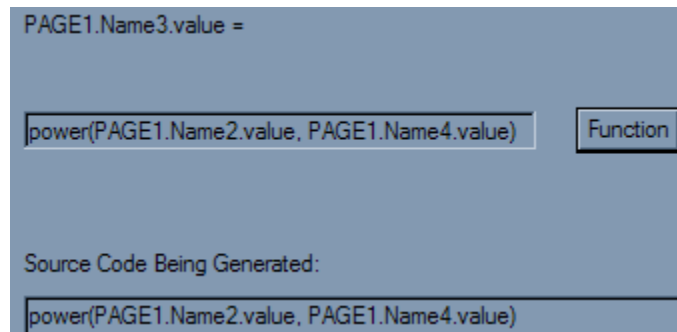
#### Method Call

**power**(number, power)

Where the number in **number** is raised to the power in **power**.

1. Right click on the **Solution** field under the **Power** label and select **Wizards > Compute Wizard**.

2. Set the **Property to Set** parameter to **Value**. Select the radio button '**The value is equal to a function**' and then click **Next >**.
3. In the next window, click on **Function**.
  - a. In the function window, choose **Power** from the drop down list.
  - b. In the Parameters section under **number**, select "**Choose an item on the form**", Select the hand icon, Click **Number** field under the **Power** label.
  - c. In the Parameters section under **power**, select "**Choose an item on the form**", Select the hand icon, Click **Power** field under the **Power** label.
  - d. Click **OK** to return to the Compute Wizard.
  - e. Your compute wizard should look like the screen shot below.



4. Click **Finish** to complete the compute.
5. View the form in the **Preview** mode to test the form. Enter a number and a power and click **Solution** field.

## power

This exercise uses the power function to return the number represented in number

Number	Power	Solution
<input type="text" value="12"/>	<input type="text" value="2"/>	<input type="text" value="144"/>

### Function 5 – Log

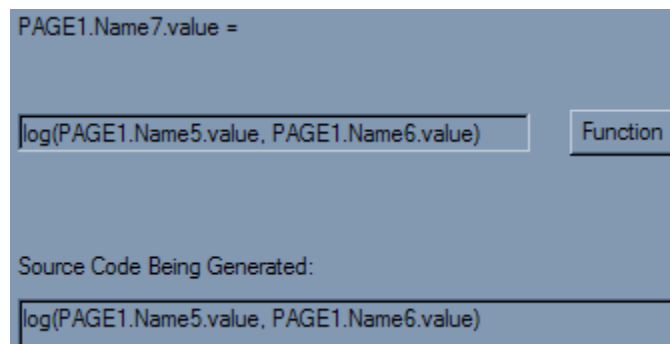
This exercise uses the log function to return the logarithm of the number represented in *number* to the base indicated by *base*.

#### Method Call

**log**(number, base)

Where the number in **base** is the base for which the logarithm will be computed.

1. Right click on the **Solution** field under the **Log** label and select **Wizards > Compute Wizard**.
2. Set the **Property to Set** parameter to **Value**. Select the radio button '**The value is equal to a function**' and then click **Next >**.
3. In the next window, click on **Function**.
  - a. In the function window, choose **Log** from the drop down list.
  - b. In the Parameters section under **number**, select "**Choose an item on the form**", Select the hand icon, Click **Number** field under the **Log** label.
  - c. In the Parameters section under **base**, select "**Choose an item on the form**", Select the hand icon, Click **Base** field under the **Log** label.
  - d. Click **OK** to return to the Compute Wizard.
  - e. Your compute wizard should look like the screen shot below.



4. Click **Finish** to complete the compute.
5. View the form in the **Preview** mode to test the form. Enter a number and a base and click **Solution** field.

## log

This exercise uses the log function to return the logarithm of the number represented in number to the base indicated by base.

Number	Base	Solution
10	2	3.32192809

**Function 6 – substr**

This exercise uses the log function to return the substr to return the substring from the position indicated in start through end.

**Method Call**

**substr**(string, start, end)

This exercise uses the substr function to return the substring of string from the position indicated in start through end (the first character in string is zero). If start is less than zero then the substring will begin on the first character of string. If end is greater than or equal to the length of string then the substring will end on the last character of string.

An error occurs if start is greater than end, if either of start and end is not a valid integer, or if string is empty.

1. Right click on the **Substring** field under the **subStr** label and select **Wizards > Compute Wizard**.
2. Set the **Property to Set** parameter to **Value**. Select the radio button '**The value is equal to a function**' and then click **Next >**.
3. In the next window, click on **Function**.
  - a. In the function window, choose **substr** from the drop down list.
  - b. In the Parameters section under **string**: select "**Choose an item on the form**", select the hand icon, Click **String** field under the **subStr** label.
  - c. In the Parameters section under **start**: select "**Choose an item on the form**", select the hand icon, Click **Start** field under the **subStr** label.
  - d. Finally, in the Parameters section under **end**: select "**Choose an item on the form**", select the hand icon, Click **End** field under the **subStr** label.
  - e. The Function window should look as follows at the end of this step:

- f. Click **OK** to exit the function window.

4. Your compute wizard should look like the screen shot below.
5. Click **Finish** to complete the compute.
6. View the form in the **Preview** mode to test the form. Enter a String, a start and end index and click on the **Substring** field.

**Function 7 – strstr**

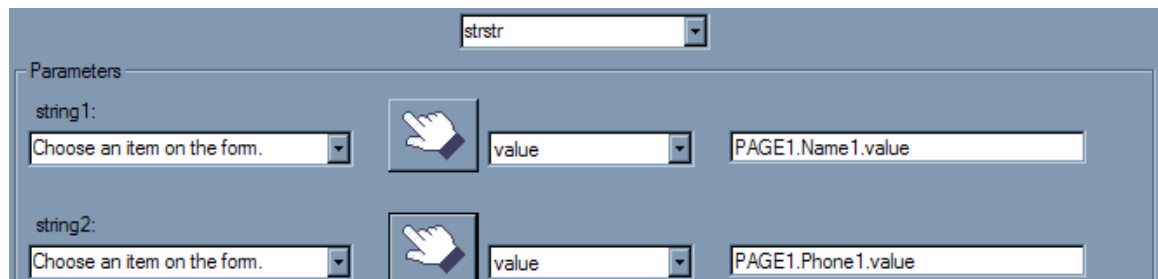
This exercise uses the strstr function. It will search for a substring within a string and return the starting location of the first occurrence or "-1" if no occurrence is found.

**Method Call**

**strstr**(string1, string2)

Where **string1** is the string and **string2** is the substring that is being looked for in **string1**.

1. Right click on the **String index** field under the **strStr** label and select **Wizards > Compute Wizard**.
2. Set the **Property to Set** parameter to **Value**. Select the radio button '**The value is equal to a function**' and then click **Next >**.
3. In the next window, click on **Function**.
  - a. In the function window, choose **strstr** from the drop down list.
  - b. In the Parameters section under **string1**: select "**Choose an item on the form**",
  - c. Select the hand icon
  - d. Click **String** field under the **strStr** label.
  - e. In the Parameters section under **string2**: select "**Choose an item on the form**".
  - f. Select the hand icon
  - g. Click **Search String** field under the **strStr** label.
  - h. The Function window should look as follows at the end of this step:



4. Click **OK** to exit the function window.
5. Your compute wizard should look like the screen shot below.

PAGE1.Phone2.value =

Source Code Being Generated:

6. Click **Finish** to complete the compute.
7. View the form in the **Preview** mode to test the form.
8. Enter a String, a start and end index and click on the **Substring** field.

**strStr**

This exercise uses the strStr function. It will search for a substring within a string and return the starting location of the first occurrence or "-1" if no occurrence is found.

String

Search String      String Index  
     

### Function 8 – date

This function returns a date in "yyyymmdd" format. Either converts a number of seconds (from 12 am, January 1, 1970) or returns the current date if no value is provided. This exercise uses a button to put the current date into the field.

### Method Call

**date**(datesecs)

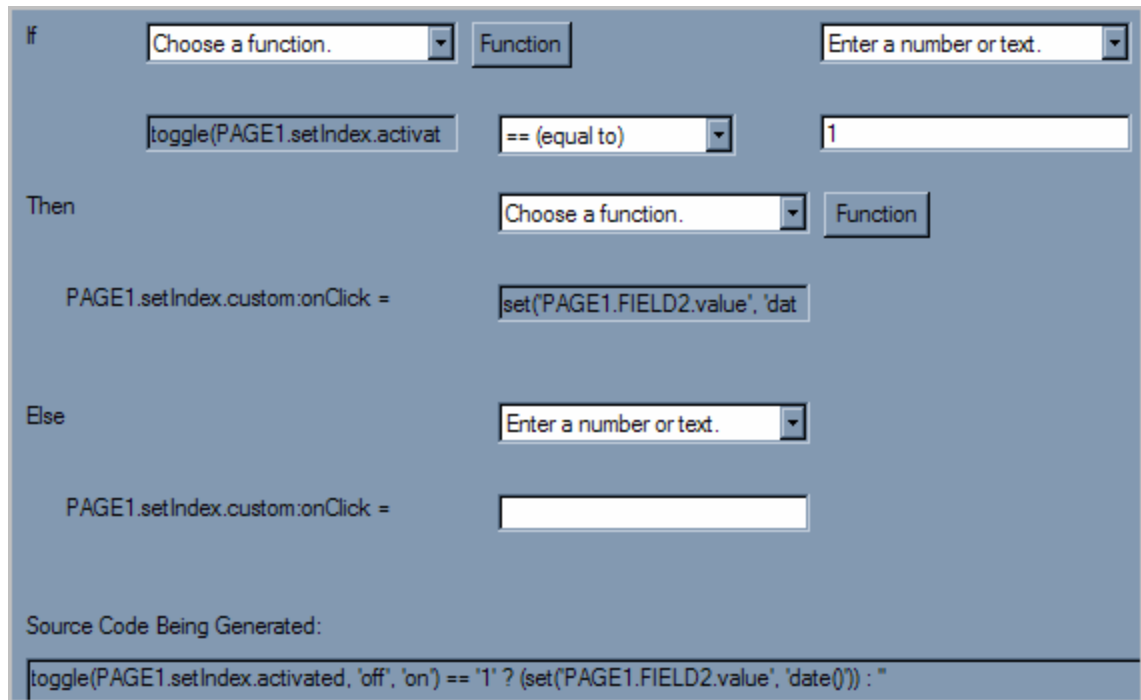
Where **datesecs** is the number of seconds from 12 am, January 1, 1970. We will leave this option blank to display the current date.

1. Right click **Today's Date** button (under the time label) and select **Wizards > Compute Wizard**.

2. Create a new custom option called 'onClick'.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
4. For the If statement:
  - a. In the first drop down, select '**Choose a function.**'
  - b. Click the **Function** button. A new window is opened 'Source Code Being Generated'.
  - c. In the drop down, select **Toggle**.
  - d. In the Parameters section under references, select "**Choose an item on the form**".
  - e. Select the hand icon.
  - f. Click **Today's Date** button and click **Finish**.
  - g. Select **activated** in the drop down next to the hand icon.
  - h. In the **Start** section, choose "**Enter a number or text**" and enter "off" in the text box.
  - i. In the **End** section, choose "**Enter a number or text**" and enter "on" in the text box.
  - j. Select **OK**.
  - k. Select "**Enter a number or text**" in the 2<sup>nd</sup> If drop down and enter the value of **1**.
5. In the Then drop down box, select '**Choose a function.**' And select the **Function** button.
  - a. A new window has been created. In the drop down box, select the function '**set**'.
  - b. In the **reference** parameter drop down, select "**Choose an item on the form**".
  - c. Select the hand icon,
  - d. Click **Current Date** field.
  - e. In the **value** drop down, select "**Enter a number or text**", Enter **date()** in the text field.
  - f. Click **Ok** to close the window.



- g. The compute must look like this at the end of this step



Source Code Being Generated:

```
toggle(PAGE1.setIndex.activated, 'off', 'on') == '1' ? (set('PAGE1.FIELD2.value', 'date')) : ''
```

- h. Click **Finish** to exit the compute wizard.
6. Select the **Today's Date** button in the design view and then select the **Source** view.
  7. Remove quotations if any from the **date()** function. The final code should look like:
 

```
<custom:onClick xfdl:compute="toggle (PAGE1.setIndex.activated, 'off', 'on') == '1' ? (set ('PAGE1.FIELD2.value', date ())) : ''"></custom:onClick>
```
  8. To test the form click on the **preview** tab. Click the **Today's Date** button.

## date

This exercise uses a button to put the current date into the field.

**Current Date**

**Function 9 – time**

This exercise uses a button to put the current time in the field in "hh:mm AM" format.

**Method Call**

**time()**

1. Right click **Today's Time** button (under the time label) and select **Wizards > Compute Wizard**.
2. Create a new custom option called 'onClick'.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
4. For the If statement:
  - a. In the first drop down, select '**Choose a function.**'
  - b. Click the **Function** button. A new window is opened 'Source Code Being Generated'.
  - c. In the drop down, select **Toggle**.
  - d. In the Parameters section under references, select "**Choose an item on the form**", Select the hand icon, Click **Today's Time** button and click **Finish**.
  - e. Select **activated** in the drop down next to the hand icon.
  - f. In the **Start** section, choose "**Enter a number or text**" and enter "off" in the text box.
  - g. In the **End** section, choose "**Enter a number or text**" and enter "on" in the text box.
  - h. Select **OK**.
  - i. Select "**Enter a number or text**" in the 2<sup>nd</sup> If drop down and enter the value of **1**.
5. In the Then drop down box, select '**Choose a function.**' And select the **Function** button.
  - a. A new window has been created. In the drop down box, select the function '**set**'.
  - b. In the reference parameter drop down, select "Choose an item on the form", Select the hand icon, Click Current **Time** field. In the **value** drop down, select "**Enter a number or text**", Enter **time()** in the text field. Click **OK** to close the window. The compute must look like this at the end of this step.

If

== (equal to)

Then

PAGE1.setIndex2.custom:onClick =

Else

PAGE1.setIndex2.custom:onClick =

Source Code Being Generated:

```
toggle(PAGE1.setIndex2.activated, 'off', 'on') == '1' ? (set('PAGE1.FIELD1.value', time())) :
```

6. Click **Finish** to exit the compute wizard.
7. Select the **Today's Time** button in the design view and then select the **Source** view.
8. Remove quotations if any from the **time()** function. The final code should look like:

```
<custom:onClick xfdl:compute="toggle (PAGE1.setIndex2.activated,
'off', 'on') == '1' ? (set ('PAGE1.FIELD1.value', time ())) :
''"></custom:onClick>
```

9. To test the form click on the **preview** tab. Click the **Today's Time** button.

## time

This exercise uses a button to put the current time in the field in "hh:mm AM" format.

**Current Time**

## 3.12. Events

The XFDL Event Model tracks user actions, such as mouse movements and keyboard input. These actions are called *events*, and you can design your forms so that these events are captured and trigger specific responses. For example, using the Event Model you could:

- Create a button that changes color when the user moves the mouse over it.
- Create a form that submits when the user types F2 on the keyboard.
- Create a “help” label that displays a different message depending on which field the user is filling out.
- Create links to websites or other documents that change color once they have been clicked.

The Event Model relies on the following XFDL options:

*value* – Detects whether or not the value of an item has been changed. A value is only re-evaluated when the focus leaves the field (i.e. tabbing out of the field).

*activated* - Detects whether or not an item, page, or form has been activated by the user. For example, a button becomes activated when the user clicks it.

*dirtyflag* - Records whether the form has been updated since the last save or submission.

*focused* - Detects whether an item, page, or form currently has the input focus. For example, a field has the focus when the cursor is in it.

*focuseditem* - Specifies which item in the page currently has the focus.

*keypress* - Stores the last keystroke made by the user.

*mouseover* - Detects whether the mouse pointer is currently positioned over an item or page.

*printing* - Indicates whether the form is currently printing.

These are special options in the XFDL language that are never written out in XFDL forms. Instead, these options are created in memory only, and can be thought of as *virtual options* - they never appear in the XFDL text for a form, but are always created and maintained when the form is run.

This means that you never have to add these options to your form. You can simply assume that they will exist when the form is in use. Furthermore, each option is automatically included in every item that supports it. For example, every button item will include an *activated* option.

### 3.13. XFDL Exercise 7 – Using XFDL Events

Events are an integral part of the XFDL language. Events play a major role as they are often used by a compute to determine whether the compute is run or is ignored.

This exercise will provide:

- a brief overview of what each event does
- step by step instructions to implement the XFDL events
- examples of using the events found within XFDL

Workplace Forms Designer 2.6 is required for this exercise.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Switch to the Resource Perspective and open **XFDL\_Ex07\_XFDLEvents.xfdl** from the **TrainingExercises\_XFDL** project in the Navigator View.

#### Dirty Flag

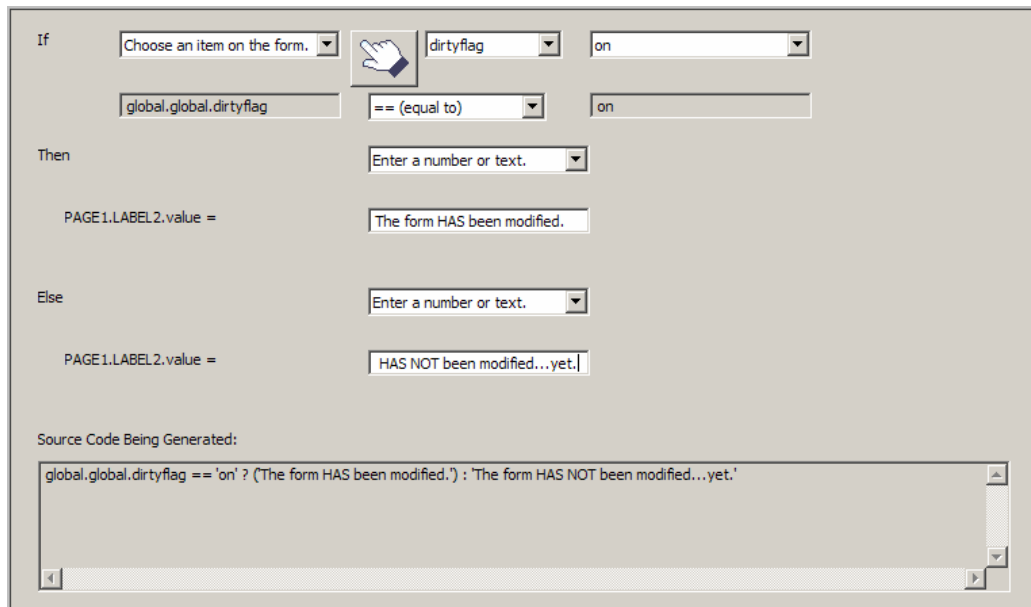
The dirtyflag is a global variable that records whether the form has been updated since the last save or submission. If the user attempts to close the form when the dirtyflag is set to on, the user will first be prompted to save their changes.

In this exercise, the compute wizard will be used, but the code view of the compute will look similar to the compute below.

```
<value compute="global.global.dirtyflag == 'on' ? 'The form HAS
been modified.' : 'The form HAS NOT been
modified...yet.'"></value>
```

1. Right click LABEL2 (under the dirtyflag label) and select **Wizards > Compute Wizard**
2. At 'Property to Set' select **value** from the popup.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
4. For the If statement:
  - a. In the first popup, select '**Choose an item on the form.**'
  - b. Click the hand icon. In the Outline View, expand the globalpage and select **Form Global**

- c. In the Compute Wizard Dialog, a new drop down menu will appear with the default of activated. Click the drop down and select '**dirty flag**'.
  - d. On the last drop down on the 'If' line select '**on**'
5. At this point, the Source Code Being Generated should look like this:
- ```
global.global.dirtyflag == 'on' ? ('') : ''
```
6. The 'Then' drop down should contain 'Enter a number or text'. If this is not the case click the arrow on the drop down and select '**Enter a number or text.**'
    - a. In the field below, type **The form HAS been modified.**
  7. The 'Else' drop down should contain 'Enter a number or text'. If this is not the case click the arrow on the drop down and select '**Enter a number or text.**'
    - a. In the field below, type **The form HAS NOT been modified...yet.**
8. Your compute wizard should look like the screen shot below.



The screenshot shows the Compute Wizard dialog with the following configuration:

- If:** Choose an item on the form. (dropdown) **dirtyflag** (dropdown) **on** (dropdown)
- global.global.dirtyflag** (text field) **== (equal to)** (dropdown) **on** (dropdown)
- Then:** Enter a number or text. (dropdown)
- PAGE1.LABEL2.value =** **The form HAS been modified.** (text field)
- Else:** Enter a number or text. (dropdown)
- PAGE1.LABEL2.value =** **HAS NOT been modified...yet.** (text field)
- Source Code Being Generated:**

```
global.global.dirtyflag == 'on' ? ('The form HAS been modified.'): 'The form HAS NOT been modified...yet.'
```

9. Select **Finish**
10. Select the preview tab to test your form.

### Activated

Specifies whether an item, page or form is currently activated by the user or not. The activated event is usually associated when using a compute toggle within a button. The activated option is also used with actions, cells, combo boxes, popups, page global and form global items.

In this exercise, you will create a custom compute that will notify when the item has been activated by throwing a viewer messageBox with an appropriate message. We will be creating the custom compute using the compute wizard. The code view of our compute will look like the compute below:

```
<custom:onClick xfdl:compute="
  toggle(activated, 'off', 'on') == '1'
  ? viewer.messageBox('You just activated/clicked the button!',
    'activated event')
  : '' "></custom:onClick>
```

1. Right click BUTTON1 (under the activated label) and select **Wizards > Compute Wizard**
2. Select the button Create Custom Option
3. Use the namespace **custom** and input into Option Name: **onClick**, then select **Ok**.
4. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.

a. For the If statement:

- i. In the first drop down, select '**Choose a function.**'
- ii. Click the **Function** button. A new window is opened 'Source Code Being Generated'.

b. In the drop down, select **Toggle**.

c. In the Parameters section under references, select "**Choose an item on the form**", Select the hand icon, Click **BUTTON1** and click **Finish**.

d. Select **Activated** in the drop down next to the hand icon.

e. Select "**Enter a number or text**" in the start parameter and enter the value **off**.

f. Select "**Enter a number or text**" in the end parameter and enter the value **on**.

g. Select **OK**.

h. Select "**Enter a number or text**" in the 2<sup>nd</sup> If drop down and enter the value of **1**.

5. At this point, the Source Code Being Generated should look like this:

```
toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1' ? (') : ''
```

6. In the Then drop down box, select '**Choose a function.**' And select the **Function** button.

7. A new window has been created. In the drop down box, select the function '**viewer.messageBox**'

8. In the message parameter drop down, select '**Enter a number or text**' and enter the value **You just activated/clicked the button!**
9. In the caption parameter drop down, select "**Enter a number or text**' and enter the value **activated event**.
10. In the messagetype parameter drop down, select "**Enter a number or text**' and enter the value **0**.
11. Select **OK**.
12. In the Else drop down box, select '**Enter a number or text**.' And leave the value empty.
13. Your compute wizard should look like the screen shot below.

The screenshot shows the 'Compute Wizard' dialog box with the following configuration:

- If:**
  - Function: Choose a function. (dropdown)
  - Function: toggle(PAGE1.BUTTON1.activated, 'off', 'on')
  - Operator: == (equal to) (dropdown)
  - Value: 1 (input field)
- Then:**
  - Function: Choose a function. (dropdown)
  - Function: viewer.messageBox('You just activated/clicked the button', 'activated event', '0')
- Else:**
  - Function: Enter a number or text. (dropdown)
  - Function: viewer.messageBox('You just activated/clicked the button', 'activated event', '0')
- Source Code Being Generated:**

```
toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1' ? (viewer.messageBox('You just activated/clicked the button', 'activated event', '0')) : "
```

14. Select **Finish**.
15. Select the preview tab to test your form.

### Mouseover – Part I

The mouseover event specifies whether the mouse pointer is currently over an item or page. When the event occurs, XFDL can indicate this event by changing the color of the item or by displaying a different value.

Our first example, we will create a mouse over event within the value tag of an item using the compute wizard. The end compute should look like this:

```
<value compute="
```



```
toggle(mouseover, 'off', 'on') == '1'
? 'Congratulations!'
: 'Mouse over me!' ">Mouse over me!</value>
```

1. Right click **FIELD3** (under the mouseover label) and select **Wizards > Compute Wizard**
2. Select **value** in the drop down.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.

- a. For the If statement:
  - i. In the first drop down, select '**Choose a function.**'
  - ii. Click the **Function** button. A new window appears called 'Source Code Being Generated'.
- b. In the drop down, select **Toggle**.
- c. In the Parameters section under references, select '**Choose an item on the form**',
- d. Select the hand icon
- e. Click **FIELD3**
- f. Click **Finished**.
- g. Select **mouseover** in the drop down next to the hand icon.
- h. Select '**Enter a number or text**' in the start parameter and enter the value **off**.
- i. Select '**Enter a number or text**' in the end parameter and enter the value **on**.
- j. Select **OK**.
- k. Select "**Enter a number or text**' in the 2<sup>nd</sup> If drop down and enter the value of **1**.

4. At this point, the Source Code Being Generated should look like this:

```
toggle(PAGE1.FIELD3.mouseover, 'off', 'on') == '1' ? ('') : ''
```

5. Open the **Then** drop down box and select '**Enter a number or text.**'
  - a. In the field below, type **Congratulations!**
6. Open the Else drop down box and select '**Enter a number or text.**'

- a. In the field below, type **Mouse over me!**
7. Your compute wizard should look like the screen shot below.

8. Select **Finish**.
9. Select the preview tab to test your form.

### Mouseover – Part II

Our next example, we will create a mouse over event with the bgcolor tag of an item using the compute wizard. The end compute should look like this:

```
<bgcolor compute="
  toggle(mouseover, 'off', 'on') == '1'
  ? '#408080' : '#FFFF80' ">#FFFF80</bgcolor>
```

1. Right click **LABEL1** (under FIELD3) and select **Wizards > Compute Wizard**
2. Select **bgcolor** in the drop down.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
  - a. For the If statement:
    - i. In the first drop down, select '**Choose a function.**'
    - ii. Click the **Function** button. A new window is opened called 'Source Code Being Generated'.

- b. In the drop down, select **Toggle**.
  - c. In the Parameters section under reference, select '**Choose an item on the form**', Select the hand icon, Click **LABEL1** and click **Finish**.
  - d. Select **mouseover** in the drop down next to the hand icon.
  - e. Under **start (optional)**, select '**Enter a number or text**' and enter the value **off**.
  - f. Under **end (optional)**, select '**Enter a number or text**' and enter the value **on**.
  - g. Select **OK**.
  - h. Select '**Enter a number or text**' in the 2<sup>nd</sup> If drop down and enter the value of **1**.
4. At this point, the Source Code Being Generated should look like this:
- ```
toggle(PAGE1.LABEL1.mouseover, 'off', 'on') == '1' ? ('') : ''
```
5. Open the '**Then**' drop down box and select '**Enter a number or text.**'
    - a. In the field below, type **#408080**
  6. Open the '**Else**' drop down box and select '**Enter a number or text.**'
    - a. In the field below, type **#FFFF80**
  7. Your compute wizard should look like the screen shot below:

The screenshot shows a compute wizard interface with the following configuration:

- If:**
  - Function: Choose a function. (Dropdown)
  - Function: Function (Text)
  - Enter a number or text. (Dropdown)
  - toggle(PAGE1.LABEL1.mouseov (Text)
  - == (equal to) (Dropdown)
  - 1 (Text)
- Then:**
  - Enter a number or text. (Dropdown)
  - PAGE1.LABEL1.bgcolor = (Text)
  - #408080 (Text)
- Else:**
  - Enter a number or text. (Dropdown)
  - PAGE1.LABEL1.bgcolor = (Text)
  - #FFFF80 (Text)
- Source Code Being Generated:**

```
toggle(PAGE1.LABEL1.mouseover, 'off', 'on') == '1' ? ('#408080') : '#FFFF80'
```

8. Select **Finish**.
9. Select the preview tab to test your form.

### Focuseditem

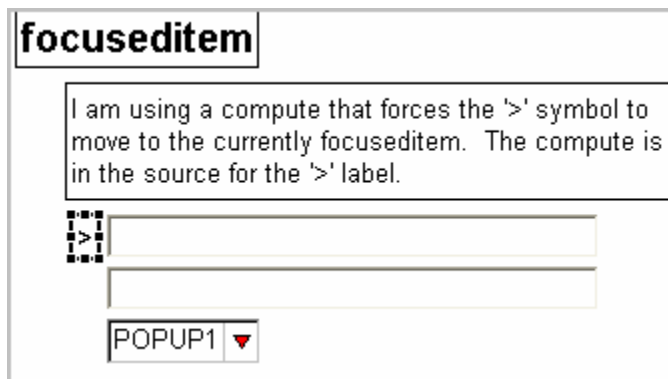
Focuseditem specifies which item in the page currently has the focus.

In LABEL12 the final compute will look like this:


```
<before compute="global.focuseditem">FIELD1</before>
```

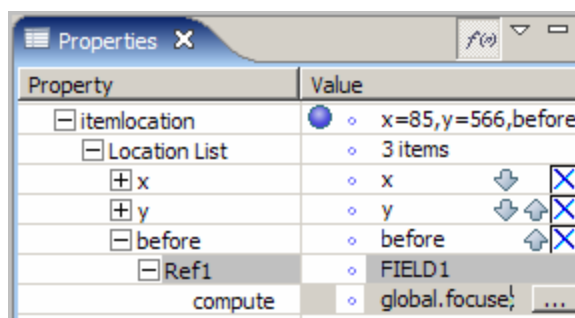
#### Create the before tag

1. Highlighting **LABEL12**
2. Hold down the Ctrl button and select another item (**FIELD1**)
3. Right click LABEL12 and select **Relative Align > Relatively Align Before**
  - a. LABEL12 should now be positioned before FIELD1, as shown below.



#### Modify the before tag

4. Highlight **LABEL12**
5. In the Properties Tab, expand itemlocation, Location List and before.
6. Highlight **Ref1** and select the compute function  button.
  - a. This will add a new row under Ref1 called **compute**



7. Enter the value **global.focuseditem** into the attribute and hit enter.
8. Select the Preview tab to test your form.

### Keypress


Contains the last keystroke made by the user in the focused item, page, or form. In this exercise, keypress will be used to activate a message box when the page/form has focus. Note that when a field has focus, only the F1-12 keys will activate the message box. We will be using the following compute in the value tag of LABEL11.

```
<value compute="(global.global.keypress != '')
? viewer.messageBox('You pressed the ' + global.global.keypress
+. ' key.', 'Keypress Event', 'Keypress Event')
: 'Press a key...'">Press a key...</value>
```

1. Right click **LABEL11** (Press a key... label) and select **Wizards > Compute Wizard**
2. Select **value** in the drop down.
3. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
  - a. For the If statement:
    - i. In the first drop down, select '**Choose an item on the form.**'
    - ii. Click the hand icon. In the Outline tab, expand the globalpage and select **Form Global.**
  - b. A new drop down menu has appeared with the default of activated. Select '**keypress**'.
  - c. Select **!= (not equal to)** in the drop down.
  - d. Leave the final **If** dropdown field empty.
4. At this point, the Source Code Being Generated should look like this:

```
global.global.keypress != '' ? ('') : ''
```
5. In the **Then** drop down box, select '**Choose a function.**' And select the **Function** button.
  - a. A new window has been created. In the drop down box, select the function '**viewer.messageBox**'
  - b. In the message parameter drop down, select '**Choose an item on the form.**'
  - c. Click the hand icon.
  - d. In the Outline tab, expand the globalpage and select **Form Global.**

- e. Select **keypress** in the drop down next to the hand icon.
  - f. In the caption parameter drop down, select '**Enter a number or text**' and enter the value **Keypress Event**.
  - g. In the messagetype parameter drop down, select '**Enter a number or text**' and enter the value **Keypress Event**.
  - h. Select **OK**.
6. In the **Else** drop down box, select '**Enter a number or text.**' And enter the value **Press a key... .**
- a. The compute window should look like the screen capture below.

If  

Then

PAGE1.LABEL11.value =

Else

PAGE1.LABEL11.value =

Source Code Being Generated:

```
global.global.keypress != " ? (viewer.messageBox(global.global.keypress, 'Keypress Event', 'Keypress Event')) : 'Press a key...'
```

7. Select **Finish**.
8. Highlight **LABEL11**.
9. Select the Source tab.
10. Edit the first parameter in viewer.messageBox. Change the parameter to '**You pressed the +. global.global.keypress +. ' key.'**
11. Select the Preview tab to test your form.

## Focused

Specifies whether an item, page, or form currently has the input's focus. The exercise will highlight all text when the FIELD4 gains focus. A custom compute will be used and the source code will look similar to the following code:

```
<custom:onFocus xfdl:compute="
  toggle(focused, 'off', 'on') == '1'
  ? viewer.setCursor('0', strlen(value))
  : ''"></custom:onFocus>
```

1. Right click FIELD4 and select **Wizards > Compute Wizard**
2. Select the button Create Custom Option
3. Use the namespace custom and input into Option Name: **onFocus**, then select **OK**.
4. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
  - a. For the **If** statement:
    - i. In the first drop down, select '**Choose a function.**'
    - ii. Click the Function button. A new window is opened called 'Source Code Being Generated'.
  - b. In the drop down, select **Toggle**.
  - c. In the Parameters section under reference, select '**Choose an item on the form**', Select the hand icon, Click FIELD4 and click **Finished**.
  - d. Select focused in the drop down next to the hand icon.
  - e. Select '**Enter a number or text**' in the start parameter and enter the value **off**.
  - f. Select '**Enter a number or text**' in the end parameter and enter the value **on**.
  - g. Select **OK**.
  - h. Select "Enter a number or text' in the 2<sup>nd</sup> **If** drop down and enter the value of 1.
5. At this point, the Source Code Being Generated should look like this:
 

```
toggle(PAGE1.FIELD4.focused, 'off', 'on') == '1' ? ('') : ''
```
6. In the **Then** drop down box, select '**Choose a function.**' And select the **Function** button.
  - a. A new window has been created. In the drop down box, select the function '**viewer.setCursor**'

- b. In the start value parameter drop down, select '**Enter a number or text.**' Enter the value **0**.
  - c. In the end value parameter drop down, select '**Enter a number or text**' and enter the value **1**.
  - d. Select **OK**.
7. In the **Else** drop down box, select '**Enter a number or text.**' And leave the value empty.
  8. Your compute wizard should not look like the screen shot below:

The screenshot shows a compute wizard interface with the following configuration:

- If:**
  - Start value parameter: Choose a function. (dropdown)
  - Function: Function (button)
  - End value parameter: Enter a number or text. (dropdown)
  - Condition: toggle(PAGE1.FIELD4.focused, == (equal to) (dropdown), 1 (text input))
- Then:**
  - Function: Choose a function. (dropdown)
  - Function: Function (button)
  - Action: PAGE1.FIELD4.custom:onFocus = viewer.setCursor('0', '1') (text input)
- Else:**
  - Function: Enter a number or text. (dropdown)
  - Action: PAGE1.FIELD4.custom:onFocus = (text input)

Source Code Being Generated:

```
toggle(PAGE1.FIELD4.focused, 'off', 'on') == '1' ? (viewer.setCursor('0', '1')) :
```

9. Select **Finish**.
10. Highlight **FIELD4**.
11. Select the Source tab.
12. Edit the second parameter in viewer.setCursor. Change the parameter to **strlen(value)**
13. Select the Preview tab to test your form.

### Value

Value reflects the contents of an item. In this exercise, when tabbing out of FIELD5, a message box will popup and display the value of the field. The compute will look similar to this;

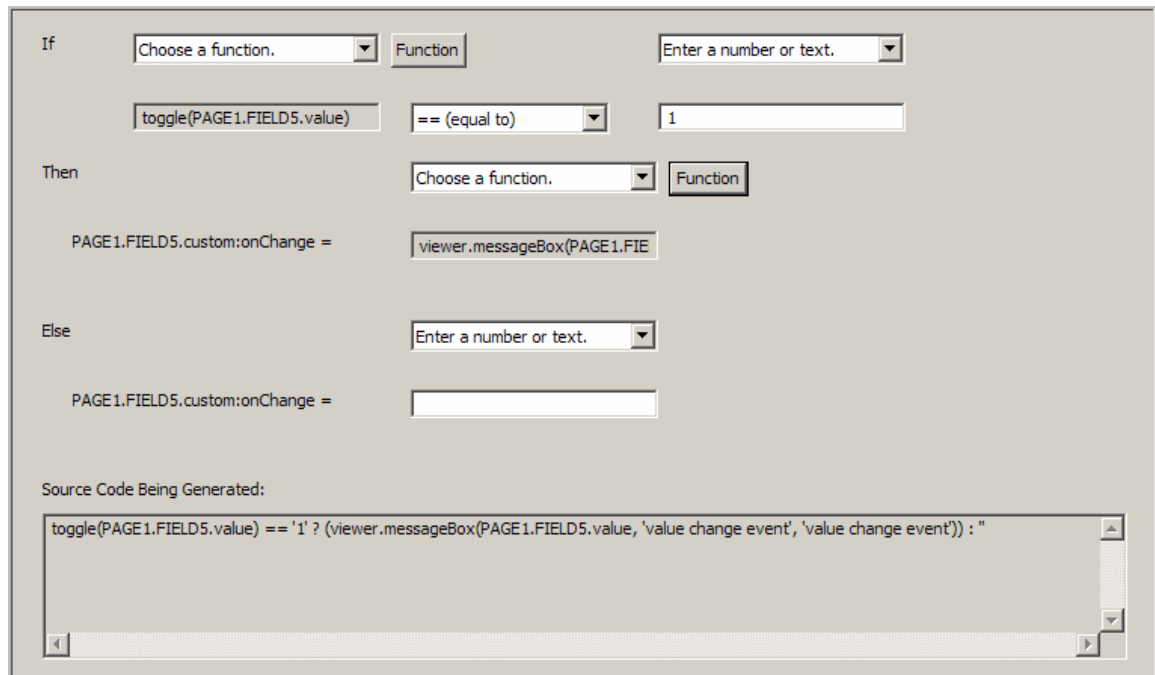


```
<custom:onChange xfdl:compute="
  toggle(value) == '1'
  ? viewer.messageBox('You just changed the fields value to
&quot;' + . value + '&quot;', 'value change event')
  : ''"></custom:onChange>
```

1. Right click FIELD5 and select **Wizards > Compute Wizard**
2. Select the button Create Custom Option
3. Use the namespace custom and input into Option Name: **onChange**, then select **OK**.
4. Select the radio button 'The value is determined by a decision (If/Then/Else)' and then click **Next >**.
  - a. For the **If** statement:
    - i. In the first drop down, select '**Choose a function.**'
    - ii. Click the Function button. A new window is opened called 'Source Code Being Generated'.
  - b. In the drop down, select **Toggle**
  - c. In the Parameters section under references, select '**Choose an item on the form**'
  - d. Select the hand icon
  - e. Click **FIELD5**
  - f. Click **Finished**
  - g. Select value in the drop down next to the hand icon
  - h. Leave the start parameter empty
  - i. Leave the end parameter empty
  - j. Select **OK**
  - k. Select '**Enter a number or text**' in the 2<sup>nd</sup> If drop down and enter the value of '1'
5. At this point, the Source Code Being Generated should look like this:

```
toggle(PAGE1.FIELD5.value) == '1' ? ('') : ''
```
6. In the **Then** drop down box, select '**Choose a function.**' And select the **Function** button.
  - a. A new window has been created. In the drop down box, select the function '**viewer.messageBox**'
  - b. In the message parameter drop down, select '**Choose an item on the form**'

- c. Click the hand icon. The form canvas will be brought to the forefront.
  - d. Select **FIELD5**. If after clicking the field the compute dialog does not come back then click **Finished** (located in the top left corner).
  - e. In the drop down next to the hand icon, select **value**.
  - f. In the caption parameter drop down, select '**Enter a number or text**' and enter the value **change event**.
  - g. In the messagetype parameter drop down, select '**Enter a number or text**' and enter the value **change event**.
  - h. Select **OK**.
7. In the **Else** drop down box, select '**Enter a number or text.**' and leave the value empty.
  8. Your compute wizard should not look like the screen shot below:



The screenshot shows a 'Compute Wizard' dialog box with the following configuration:

- If:** Dropdown: 'Choose a function.', Button: 'Function', Dropdown: 'Enter a number or text.', Expression: 'toggle(PAGE1.FIELD5.value) == (equal to) 1'
- Then:** Dropdown: 'Choose a function.', Button: 'Function', Expression: 'PAGE1.FIELD5.custom:onChange = viewer.messageBox(PAGE1.FIE'
- Else:** Dropdown: 'Enter a number or text.', Expression: (empty)
- Source Code Being Generated:** `toggle(PAGE1.FIELD5.value) == '1' ? (viewer.messageBox(PAGE1.FIELD5.value, 'value change event', 'value change event')) : ''`

9. Select **Finish**.
10. Highlight **FIELD5**.
11. Select the Source tab.
12. Edit the first parameter in viewer.messageBox. Change the parameter to '**You just changed the fields value to &quot; + value + '&quot;**'
13. Select the Preview tab to test your form.

## 3.14. Custom Options

Because XFDL is an XML document it is possible to create your own “custom” elements. Any element that is created by the user that is not part of the XFDL schema is considered a custom option. Custom options allow you to add application specific information to your form. There are two reasons for creating custom options:

- Creating re-usable variables
- Containing computes

### 3.14.1. Creating Re-Usable Variables

Nearly every programming language has a variable construct. Variables are useful for storing data that maybe used in multiple places within an application. Variables can be used in mathematical calculations, string concatenation, or data extraction (i.e. placing a repeated value in a variable that can be referenced). Since XFDL is a programming language it also has a variable construct. Variables in XFDL are “custom” XML elements created in a “custom” namespace. Once your namespace has been created you can create any elements that you want and these elements can be used as re-usable variables.

An example of a variable is an element created in the default custom namespace, that is defined when you create an XFDL form in the Designer, to keep track of a row count that increments as rows are added to a duplicating section.

```
<custom:rowCount>5</custom:rowCount>
```

An example of a compute that references this variable is:

```
<label sid="rowCount_lbl">
  <value compute="`There are' + . custom:rowCount + ` rows.'"
</label>
```

When you reference the custom option “custom:rowCount” within a compute the Viewer will first retrieve the value of the custom option, which is ‘5’, and then continue evaluating the compute. The above compute will set the value of the label to “There are 5 rows”.

### 3.14.2. Creating Containers for Computes

There are times when it makes sense that your computes exist within the option that they are dynamically determining. For example computes that determine an item’s visibility, active state or required status are perfect candidates. However sometimes computes cannot be placed inside the option that they manipulate. If you are dynamically determining the value of a field then your compute must be placed outside of the value option.

When text is entered into a field, the text takes precedence over any compute that may exist; in that instance of the form the compute is “over-written.” In cases where the item’s value is read-only the original rules apply (it is appropriate for the compute to exist within the value option). But when a field must remain editable and support a compute then it must be placed within a custom option that serves as a container.

The following compute will set the value of the field to its current value multiplied by 1.07.

```
<custom:setValue xfdl:compute="
  toggle(value) == '1' ? set('value', value * '1.07') : ''">
</custom:setValue>
```

If this compute was created in the `<value>` option then as soon as the user entered text in the field the compute would be removed.

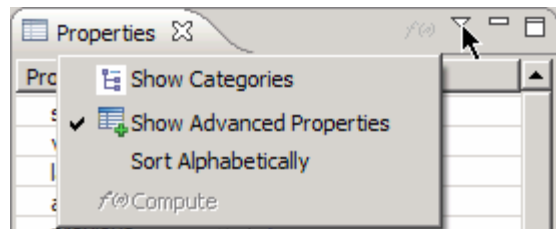
An example of another compute that requires a container is when the compute is not specifically changing an option. The most common example of this is creating a compute that performs an action when a button is clicked:

```
<custom:onClick xfdl:compute="
  toggle(activated, 'off', 'on') == '1' ? 'set(...)' : ''"
```

The above compute does not necessarily operate on a specific option and should be placed inside of a custom option.

### 3.14.3. Exercise - Creating a Custom Option in the Designer

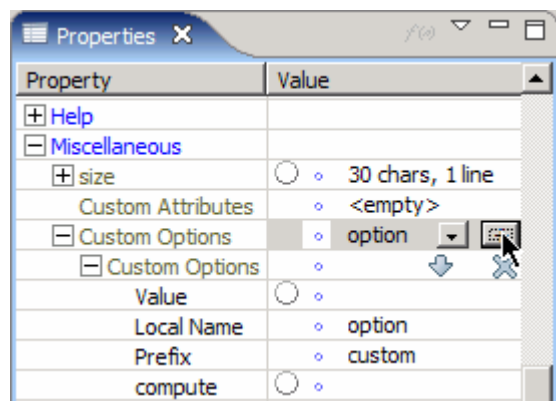
Custom Options can be added to any form item through the Properties View. Open the Properties View; make sure that the Advanced Properties are active by selecting the down arrow and Show Advanced Properties.



1. Expand the **Miscellaneous** category, select the value of the **Custom Options** property and press the add button.

2. Expand **Custom Options > Custom Options** and now you can set the properties of the custom option that you just added.

- The **Value** is the content of the custom option
- The **Local Name** is the option name (i.e. `<localName></localName>`)



- The **Prefix** refers to the namespace that the custom option will belong to; all custom options must be assigned a namespace. By default you will be able to select either the custom or designer namespace.
- The **compute** property contains a conditional expression which can be built using the compute wizard. Select the value of the compute property and a “...” button will appear that opens the compute wizard.

## 4. Advanced XFDL Concepts

### 4.1. Audience

This section is geared toward applying some of the concepts learned in “Understanding XFDL”.

### 4.2. Overview

XFDL is a rich language that encompasses presentation, logic and data management. Building dynamic forms is somewhat of an art form that does take practice. Unfortunately the Designer is still in its infancy and building the more complex forms requires a deeper understanding of how XFDL works. In the future we hope the Designer will abstract that complexity but until then we must explore deeper into XFDL.

The objectives of this module are:

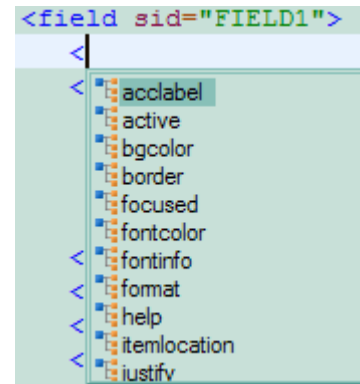
- The reader will be able to use the Source View to write XFDL code; leveraging content assist and other tools.
- The reader will be able to create a form that contains a dynamically duplicating section (i.e. pages or items).

### 4.3. Working with the Source View

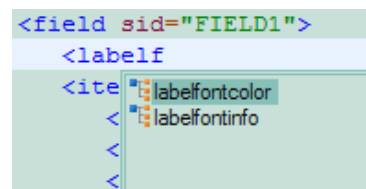
#### 4.3.1. What is Code Completion?

Code completion, also known as content assist, is a function that interprets what is being entered and provides a list of possible options to help expedite the manual coding process. In Code View if I enter an open angle bracket then the editor provides the following:

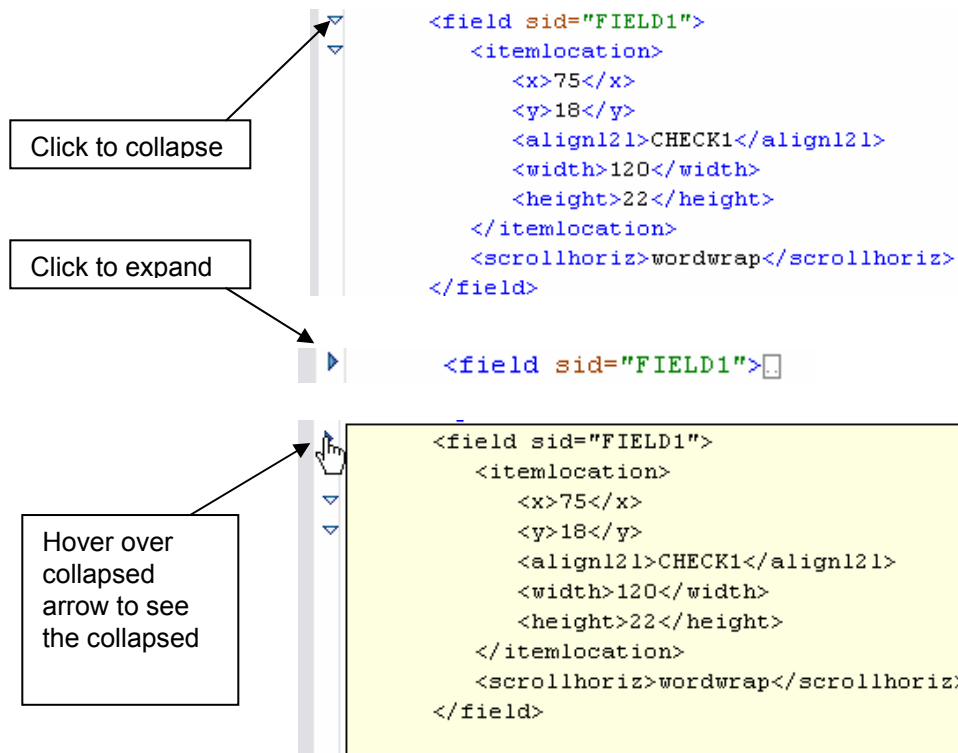
The Designer will only list the options that can legally be added to the current item. As you continue to type the list will automatically be reduced only displaying the options that still meet that criteria.



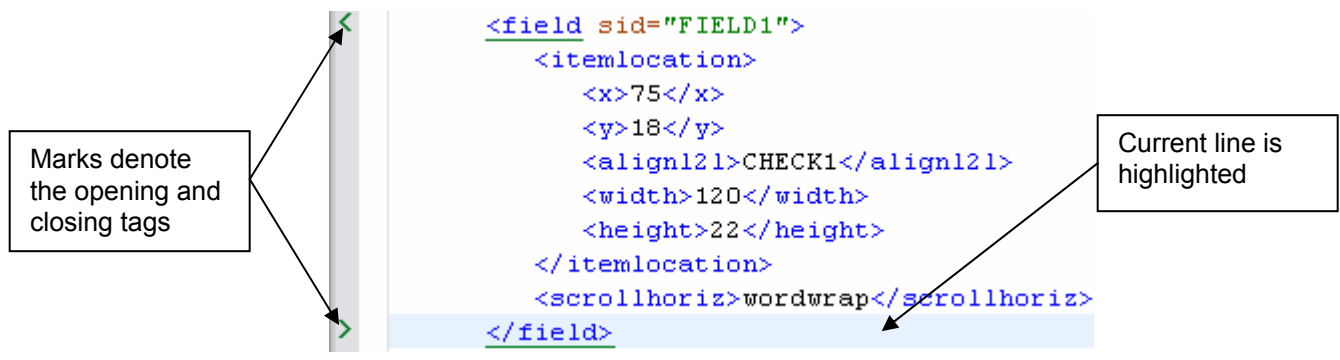
If the code completion tool does not appear or it disappears for any reason it can be re-opened by using the default key sequence of `ctrl + space bar`.



### 4.3.2. What is folding text?

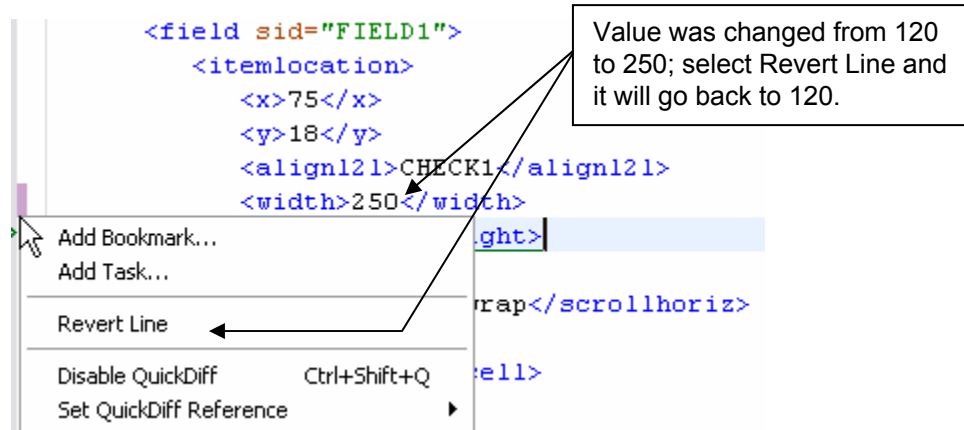


### 4.3.3. What do the marks on the left indicate? (That appear when editing)



### Reverting text back to previous value

Code View keeps track of lines as they are changing, before the document is saved. The changes are indicated by colored marks on the left margin of the code editor. Hovering over the marker will display the line before it was changed. If you want to revert back to the original text, right-click the colored mark and choose Revert Line.



### 4.3.4. What are Bookmarks?

For information about the Bookmarks View please refer to the Eclipse help, which can be found by selecting **Help > Help Contents** from the **File** menu and entering "bookmarks" as the search criteria.

### 4.3.5. Validating Your Code



This function in the Source View will validate your XFDL.

### 4.3.6. Check if code is well formed



This function in the Source View will check your code to insure that it is well formed. Any errors in format will be flagged and displayed in the Problem View.

## 4.4. Duplication

Sometimes you won't know how much information users need to enter into a form, and therefore don't know how large to make a form. Paper form designers know that users can write in the margins or on the back of a form, or even attach additional pages, but electronic form designers don't have that luxury. However, if you have a form that requires users to enter a variable amount of information, such as an accident report or a bibliography, you can create a small form that dynamically generates new pages or items as needed, instead of creating a large form with many pages or tables. This can be done using the duplicate function.

The duplicate function can duplicate items and pages. If you use it to duplicate items, it automatically duplicates the item's options and arguments. If you use it to duplicate pages, it automatically duplicates every item and option on the page.

---

Note: If you are thinking about duplicating an option...don't... you should use the set function rather than the duplicate function to duplicate options. If the option that you are setting does not exist then it will create it.

---

The duplicate function is described by 6 properties or parameters:

1. A reference to the form element that you want to duplicate.
2. The type of element referenced in parameter 1 (i.e. page, item, or option).
3. A reference to the form element that will anchor the newly duplicated element (using relative positioning) in the form's build order.
4. The type of element referenced in parameter 3.
5. The relation of the new item to the anchor item.
  - Valid choices are `append_child`, `after_sibling`, and `before_sibling`.
6. The name of the new form element.

The following code sample shows how to create a simple duplicate function:

```
duplicate('TEMPLATE_PAGE', 'page',
         'TEMPLATE_PAGE', 'page', 'before_sibling',
         'PAGE')
```

Duplicate functions are usually contained within computes that are triggered by a user's action, and are therefore often partnered with set and toggle functions, as shown below:

```
<custom:on_activated xfdl:compute=" &#xA;
toggle(activated, 'off', 'on') == '1' &#xA;
? (set('custom:page_count', custom:page_count + '1') &#xA;
+. duplicate('TEMPLATE_PAGE', 'page', 'TEMPLATE_PAGE',
'page', 'before_sibling',
```



```
        'PAGE' +. custom:page_count) &#xA;  
        +. set('PAGE' +. custom:page_count +.  
        '.pageNumber_LABEL.custom:myPage_num', page_count)) &#xA;  
        : '''>  
</custom:on_activated>
```

## 4.5. Page Duplication

You can use the *duplicate* function to duplicate entire pages. This allows the user to add an unknown or infinite number of pages to the form. This is generally done by creating a *template* of the page you want to duplicate. A template page is a page that the user can't access and is only used as a source for duplication. Since you typically don't allow the user to access the template page, its items remain blank. This means that you won't have to clear any user information from the page after it is duplicated.

When duplicating a page, you usually want to place each page one after another so that the newest page has the highest page number. Your form will run faster if you place the new pages before a static (non-duplicated) page, such as your template page, rather than placing each new page after the last duplicated page. It requires more work to evaluate the SID of the last duplicated page than it does to evaluate a static page SID.

The template page should have a Next Page and Previous Page buttons, so that users can navigate through all the newly duplicated pages. However, you should configure the Next Page button to be disabled if the next page is the template page.

Both the template page and the last page of the form usually contain an Add Page button. This button triggers the page duplication when it is clicked. You can add custom options to this button that store dynamic information, such as how many pages has been added to the form. You can then reference this information to display a running page total to the user.

---

**Note:** You can simplify page navigation by using relative page tags in the code of the Next Page and Previous Page buttons, instead of setting their url option. The relative page tags are `global.pagenext` and `global.pageprevious`.

---

## 4.6. XFDL Exercise 8 – Duplicating Form Pages

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. “PageDuplication”), then click **Finish**
  - c. The Designer should automatically switch to the Designer Perspective and open the file.
3. Create the first page of the form as shown below. In the **Properties View**, change the SID to “**page\_1**”.

Add Page    Next Page

LABEL1

**Dependant Information**

Name

Birth Date

Gender  
 Female    Male

4. Create a template page which will be copied to create new pages in the form. In the Properties View, change the SID to “**template**”.

Previous Page    Next Page

**Dependant Information**

Name

Birth Date

Gender  
 Female    Male

- a. The template page has a previous page button which is not present on the first page. Pages can only be added from the first page and hence the template page does not have an Add Page button. Note that the Next Page and Previous Page buttons are of the type pagedone.
5. On PAGE1 modify the Next Page button so that it is only active *after* a new page has been added to the form. Ensure that the Next Page button does not take the user to the template page.
  - a. Right click on the button and select **Wizards > Compute Wizard**.



- b. Change the value to active and click on 'The value is determined by a decision'. Click next.

A screenshot of a dialog box titled 'Property to Set:'. The 'active' property is selected in a dropdown menu. To the right of the dropdown is a button labeled 'Create Custom Option'. Below the dropdown are six radio button options:

- The value is equal to another form item.
- The value is equal to a function.
- The value is set by a calculation of two values.
- The value is equal to the sum of multiple fields on the form.
- The value is determined by a decision (If/Then/Else).
- The value is set by a manually created formula.

- c. Add the condition `global.pagenext == template.global`, notice that both fields are of kind **Enter a number or text**.
    - d. The built in option **global.pagenext** contains the SID of the next available page. Use this in a compute on the active option.
    - e. Set then to 'off' and else to 'on'.

if

== (equal to)

Then

PAGE1.BUTTON3.active =

Else

PAGE1.BUTTON3.active =

Source Code Being Generated:

```
'global.pagenext' == 'template.global' ? ('off') : 'on'
```

- f. Click **Finish**.
  - g. Go back into the compute wizard, select **active** from the “Property To Set” popup then select the radio button “The value is set by a manually created formula” and remove the single quotes around `global.pagenext`.
 

`global.pagenext == 'template.global' ? ('off') : 'on'`
6. Repeat the above step for the **Next Page** button on the **template page**.
  7. Select the “**Add Page**” button and open the **Compute Wizard**.
    - a. Press the “**Create Custom Option**” button. Create an option in the custom namespace called “**page\_count**.” It will be used to keep track of the current number of form pages.
    - b. And click **OK**

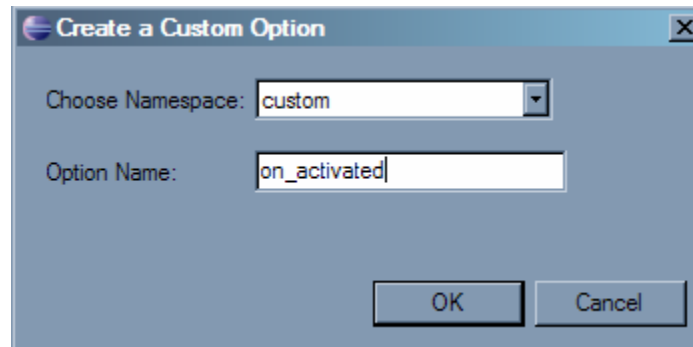
Property to Set:

**Create a Custom Option** ✕

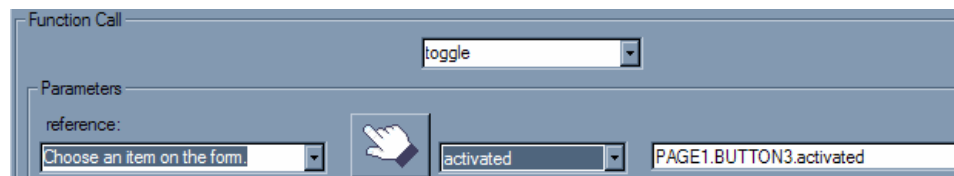
Choose Namespace:

Option Name:

- c. The custom option has been created, but we don't want to create a formula for it so just click **Cancel**. Instead we want to set the custom option's value to 1.
  - d. Select the **Add Button** and open the **Source View**, enter '1' as the value of the page\_count option.
8. Press the "**Create Custom Option**" button a second time and create an option in the custom namespace called "on\_activated." It will contain the compute that triggers the duplication. Click **OK**.



- a. Select "**The value is determined by a decision**" and click **Next >**.
- b. In the **If** section of the compute we are going to create a formula that toggles on the **activated status** of the Add Page button. Select **Choose a Function** from the popup. And press the **Function** button.
- c. Select **toggle** from the function call popup.
- d. Select **Choose an item on the form** and click the button with the hand. This will bring the form into the foreground, select the Add Button (the compute wizard should come back into focus but if it doesn't then press the finished button in the top left-hand corner).
- e. Select **activated** from the popup. Set the start and end parameters to **Enter a number or text** with values of **off** and **on** respectively.



- f. In the **Then** section of the compute select **Choose a function** and click the Function button. Select **Duplicate** from the Function Call popup.

- g. Select a **number or text** from the reference popup and enter the SID of the template page **template** in the field below it.
- h. Select a **number or text** from the type popup and enter “**page**” in the field below it.

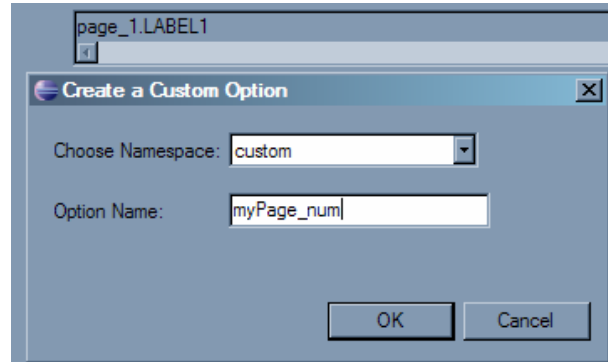
- i. Select a **number or text** from the newReference popup and enter the SID of the template page **template** in the field below it.
- j. Select a **number or text** from the newType popup and enter “**page**” in the field below it.
- k. Select a **number or text** from the location popup and enter “**before\_sibling**” in the field below it.

- l. Select a **number or text** from the newName popup and enter “**PAGE\_**” in the field below it.

9. After the page has been duplicated we need to increment the variable that contains the number of pages. Select the Add Button and Click on the Source Tab to open the Source View and change the on\_activated toggle as follows:

```
<custom:on_activated xfdl:compute="
  toggle(page_1.BUTTON1.activated, 'off', 'on') == '1'
  ? set('custom:page_count', custom:page_count + '1')
  +. duplicate('template', 'page',
              'template', 'page', 'before_sibling',
              'page_'+custom:page_count)
: ''"></custom:on_activated>
```

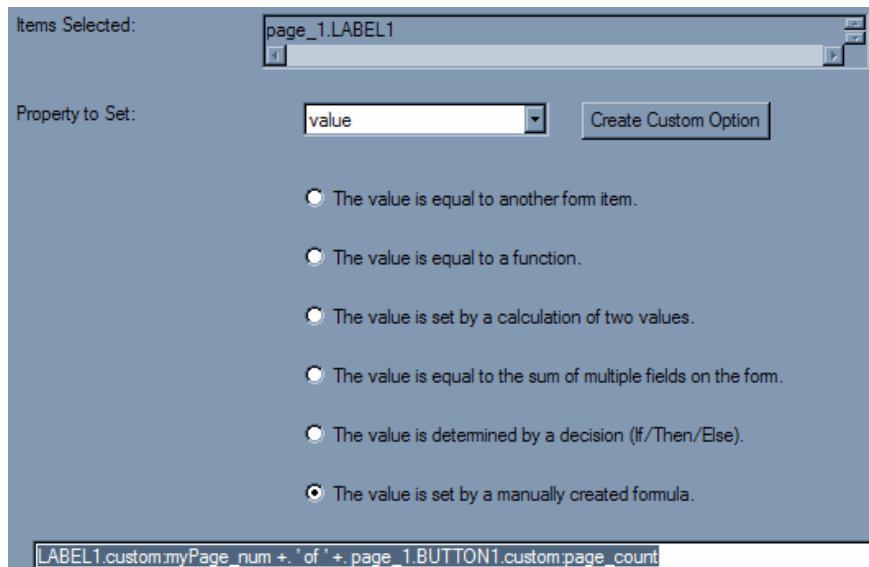
10. Create a custom option called myPage\_num in LABEL1 on both page\_1 and the template page. Select the label and open the compute wizard. Click on the **Create a Custom Option** button and create the option in the custom namespace. Click OK in the custom option dialog.



- The custom option has been created, but we don't want to create a formula for it so just click Cancel. Instead we want to set the custom option's value to 1. Select the label and open the Source View, enter '1' as the value of the myPage\_num option.
- The value of this label must be set to the current page number which is stored in custom:myPage\_num. Right click on the label and go to the compute wizard. Change the value to read

```
LABEL1.custom:myPage_num +. ' of ' +.
page_1.BUTTON1.custom:page_count
```

On the final form, this would read as '2 of 23', where the user is on the second page and there are 23 pages in total.



11. As each page gets created we want to store the current page number (BUTTON1.custom:page\_count) in the myPage\_num custom option of the label. Change the on\_activated toggle in BUTTON1 as follows:

```
<custom:on_activated xfdl:compute="
  toggle(page_1.BUTTON1.activated, 'off', 'on') == '1'
  ? (set('custom:page_count', custom:page_count + '1'))
```



```

+. duplicate('template', 'page', 'template', 'page',
            'before_sibling', 'page_'+custom:page_count)
+. set('page_' + custom:page_count +
      '.LABEL1.custom:myPage_num', custom:page_count)
: '''></custom:on_activated>

```

12. One last thing to add in the toggle is to set the url of the next page in the sequence. Change the toggle to the following

```

<custom:on_activated xfdl:compute="
toggle(page_1.BUTTON1.activated, 'off', 'on') == '1'
? (set('page_' + custom:page_count + '.BUTTON2.url',
      '#page_' + (custom:page_count + '1') + '.global')
+.set('custom:page_count', custom:page_count + '1')
+. duplicate('template', 'page', 'template', 'page',
            'before_sibling', 'page_'+custom:page_count)
+. set('page_' + custom:page_count +
      '.LABEL1.custom:myPage_num', custom:page_count))
: '''></custom:on_active>

```

Where BUTTON2 is the sid of the “Next Page” button on each page.

13. To set the Previous Page Button on each page we need to just change the template page. Open the compute wizard for the Previous Page button and change the value of the url field as follows:

Items Selected:

Property to Set:

The value is equal to another form item.  
 The value is equal to a function.  
 The value is set by a calculation of two values.  
 The value is equal to the sum of multiple fields on the form.  
 The value is determined by a decision (If/Then/Else).  
 The value is set by a manually created fomula.

14. Now preview the form and try pressing the Add Page button. To test the code that we just added, add some pages and navigate back and forth.

## 4.7. Working with the Outline View

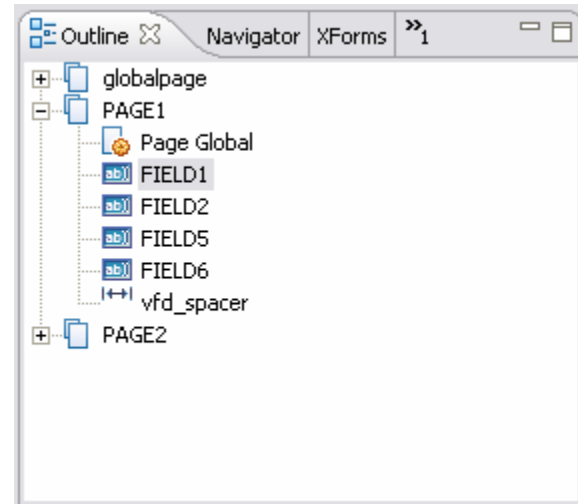
### 4.7.1. What is the Outline View?

The Outline View is a hierarchical structure that displays all the objects that make up the form. All of the items are also displayed from top to bottom in the order that they were created. This is also referred to as the “Build Order.”

When an item is selected in the Outline View it is also selected in the Design View. If an object has children (i.e. Page, etc.) then it can be expanded and collapsed like a folder in a file explorer.

The Build Order is important for a few reasons:

- When items overlap in the Design View, the last item in the Build Order will be displayed in front of the other objects.
- When positioning items relatively, an item cannot refer to an object that comes after it in the Build Order.
- The default Tab Order is defined by the Outline View or Build Order. The Tab Order defines how the cursor will move through the form’s input items when the user presses the “tab” key.



### 4.7.2. How Do I Re-arrange the Build Order?

The order of items in the Outline View can be rearranged quickly and easily. Select the item to move, click and hold the left mouse button and drag the item to its new place in the Build Order. As you drag the item a black line will appear to indicate where it will be placed.

### 4.7.3. How Do I Open the Outline View?

The Outline View can be opened from the Eclipse File Menu, select **Window > Show View > Other**. In the dialog that appears, expand **Basic** select **Outline** and click **OK**.

### 4.7.4. What Is Filter On Visible?



Filter on Visible is an option in the Outline View, when it is turned on then only the currently selected item and its children will be displayed in the Design View.

### 4.7.5. Copy/Pasting Objects in the Outline View

The Outline View supports the Cut, Copy and Paste operations. The operations can be accessed by using the standard keyboard shortcuts or by using the right-click menu. When pasting into the Outline View the object will always be appended to the end.

The Designer will automatically insure that items have unique sids by either appending a number to the item's name or incrementing it if one already exists.

#### 4.7.6. Expanding/Collapsing View

If an object has children (i.e. Page, etc.) then it can be expanded and collapsed like a folder in a file explorer. There are several different methods of performing these operations:

- Use the buttons provided in the Outline View toolbar.
- Use the “+” and “-” buttons next to the item in the Outline View.
- Use the arrow keys to navigate the Outline View (i.e. the up and down arrows move up and down the list, while the left and right arrows collapse and expand the selections respectively).

#### 4.7.7. Page Navigating



While designing a form, switching pages is done in the Outline View. Selecting a page in the Outline View displays it in the Design View. Two buttons have been provided as quick



navigation tools to cycle through the pages of the form.

## 4.8. Relative Alignment

There are two ways of aligning items within XFDL; absolute and relative. Items that are positioned absolutely will have an `<itemlocation>` that contains the x and y coordinates of the top-left corner.

```
<field sid="FIELD1">
  <itemlocation>
    <x>153</x>
    <y>89</y>
  </itemlocation>
  ...
</field>
```

Items that are positioned relatively will define a relationship to another form item. The field defined below will be positioned directly beneath "FIELD1", where the left edges are aligned and there is a 4-pixel space between the items.

```
<field sid="FIELD2">
  <itemlocation>
    <below>FIELD1</below>
  </itemlocation>
  ...
</field>
```

Relative alignment is essential when dynamically duplicating form items. Since relative positioning relies on knowing the SID of the reference object you need a way of determining it dynamically. Therefore we have provided some Viewer options that are only present when the Viewer is running (they are not serialized in the XFDL).

Dynamic Options	Description
pagenext	Returns the sid of the next page in the Build Order (from the current page)
pageprevious	Returns the sid of the previous page in the Build Order (from the current page).
itemnext	Returns the sid of the next item in the Build Order (from the current item)
itemprevious	Returns the sid of the previous page in the Build Order (from the current item)

These options are evaluated from the object that contains them. Therefore, let's re-write the above example but this time use the `itemprevious` dynamic option.

```
<field sid="FIELD2">
```

```
<itemlocation>
  <below compute="itemprevious"></below>
</itemlocation>
...
</field>
```

Since FIELD1 is the object that comes before FIELD2 in the build order the “itemprevious” option will evaluate to “FIELD1”.

## 4.9. XFDL Exercise 9 – Implementing Relative Alignment

The purpose of this example is to demonstrate how to create a dynamic form using relative alignment.

In this exercise we will:

- convert Synopsis, Author, Publisher and Date item locations to relative
- create a compute that will force the synopsis field to expand and shrink based on the amount of text entered
- create a compute that will turn the scrollbar off if the Synopsis field does not have the focus
- learn to use the Outline View to change the build order

**Book Information**

Title:

Genre:

Synopsis/Summary:

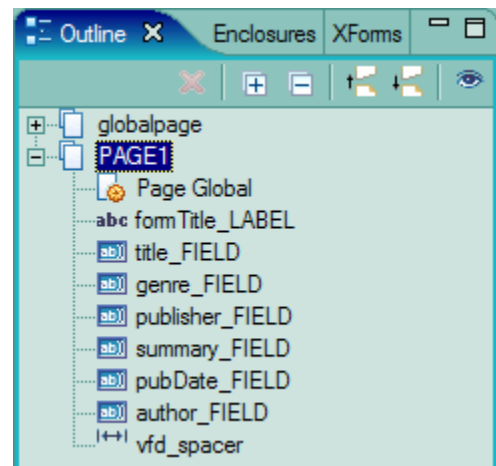
Author:

Publisher:       Date (YYYY-MM-DD):

When using absolute positioning the order is irrelevant, however with relative positioning it is essential.

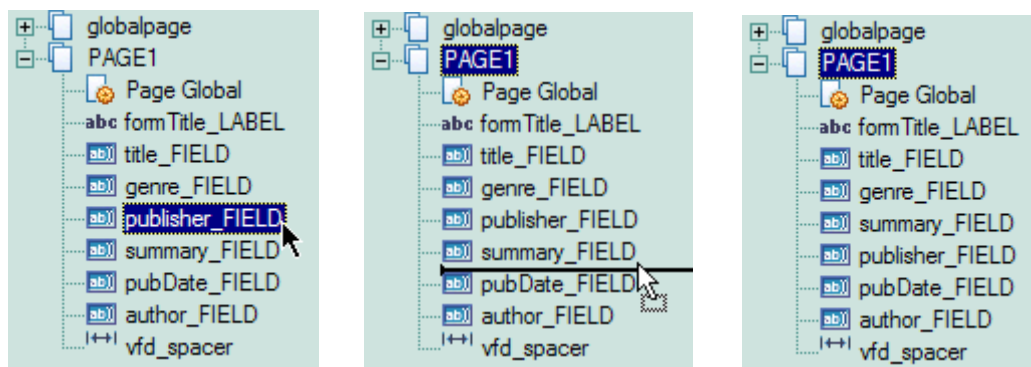
Items that are relatively positioned define a relationship and a reference item. **The reference item must precede the item being aligned.** If an item tries to reference an item that follows it in the build order then the item location will be ignored.

Since we are going to change all of the absolute item references to relative we must first make sure that the build order is correct. We need to make sure that all the items in the Outline View are in the same order as they appear on the canvas.



**Note:** The Outline View is created as items are created, so if you know that you are building a form with relative alignment if you build it in order then you will save yourself the following steps.

1. Open the exercise form file
  - a. Open Workplace Forms Designer 2.6
    - i. Start > Program Files > Workplace Forms Designer 2.6 > Designer
  - b. Switch to the Resource Perspective and open XFDL\_Ex09\_RelativePositioning.xfd from the TrainingExercises\_XFDL project in the Navigator View.
2. Modify the build order of the items in the Outline View.
  - a. The Outline View is part of the “Designer” Perspective, but if it is closed it can be opened by selecting **Window > Show View > Other...** from the Designer menu. A new dialog will appear, expand **Basic**, select **Outline** and click **OK**.
  - b. Rearrange the items so that they are in order of appearance (i.e. Synopsis, Author, Publisher and Date). Only one item in the Outline View can be adjusted at a time. Select the item, click (and hold) the left mouse button and drag it to a new location; as you drag an item a black line will appear indicating where the item would be placed if you released your left-mouse button.
  - c. Select the “publisher\_FIELD”, click (and hold) the left-mouse button and drag it between the “summary\_FIELD” and “pubDate\_FIELD” then release the mouse button.

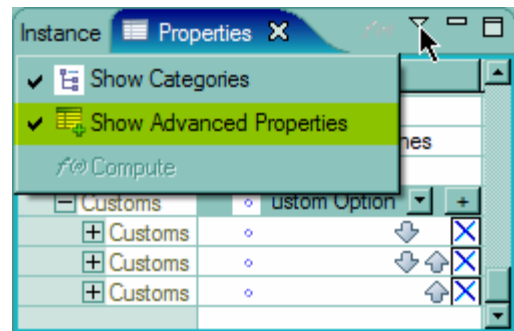


- d. Follow the same steps and move the “author\_FIELD” so that it is between “summary\_FIELD” and “publisher\_FIELD”.
3. Use the Alignment tools to convert the Synopsis, Author, Publisher and Date items to relative positioning. On the canvas:

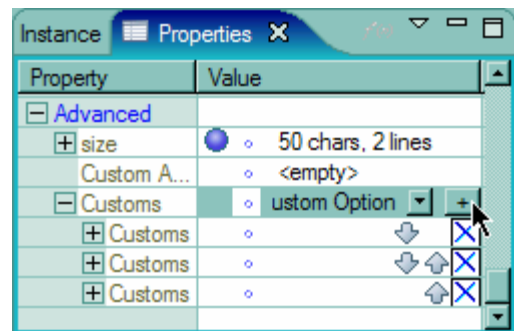


- a. Select the Author field, hold down “ctrl” and select the Synopsis field. Point your mouse at one of the selected objects, right-click and select **Relative Align > Relatively Align Below**.
  - b. Select the Publisher field, hold down “ctrl” and select the Author field. Point your mouse at one of the selected objects, right-click and select **Relative Align > Relatively Align Below**.
  - c. Select the Date field, hold down “ctrl” and select the Publisher field. Point your mouse at one of the selected objects, right-click and select **Relative Align > Relatively Align After**.
4. Add the computes that force the Synopsis field to grow and shrink.

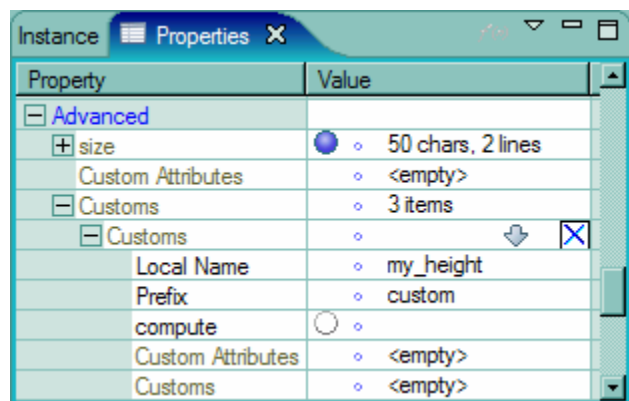
- a. Select the Synopsis field; in the Properties View make sure that the Advanced Properties are visible. They can be activated by clicking the down arrow in Properties View and selecting **Show Advanced Properties**.



- b. Once the advanced properties have been activated, create three custom options. To create a custom option, in the “Advanced” category there is an option called “Customs”, select “Custom Option” from the drop down and press the add button three times.



- c. Expand each custom item that was created and change their “Local Name” to “my\_height”, “set\_myHeight” and “min\_height”, respectively. Give “min\_height” a default value of “2”.

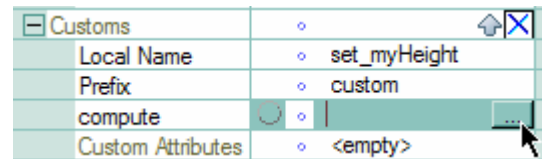


- d. The “my\_height” and “min\_height” options are variables that will be used to contain the current and minimum number of lines for the synopsis field.

- e. The “custom:set\_myHeight” option will contain the compute that will do most of the work. When the value of the synopsis field changes we want to measure the height

of its contents. We do this by using the Viewer Function called `viewer.measureHeight()`. *Please reference pg 29 of [Viewer\\_Functions.pdf](#) for more information.*

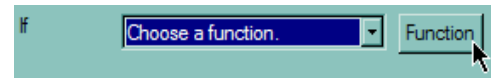
- f. Expand the **set\_myHeight** custom option, click in the value of the compute option and click the “...” button that appears.



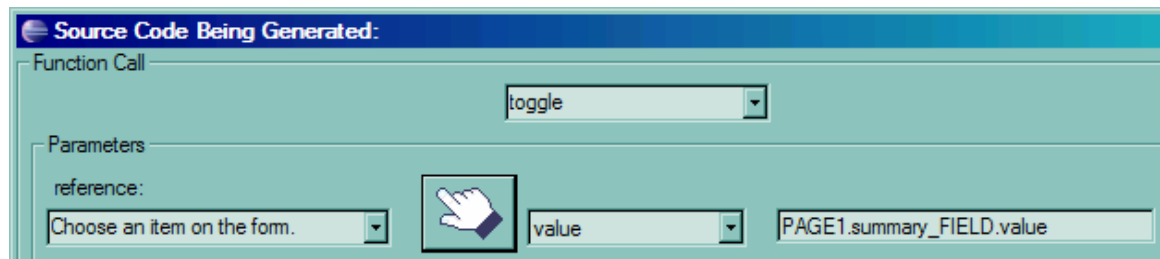
- g. Select **the value is determined by a decision (If/Then/Else)** and click **Next >**.

The value is determined by a decision (If/Then/Else).

- h. Create a toggle compute that monitors the **value** of the Synopsis field. Select **Choose a function** from the popup and press the **Function** button.



- i. In the new dialog, select toggle from the function call popup. Then select **Choose an item on the form** from the reference parameter popup and click on the button with the hand. This will bring the form canvas to the foreground, select the synopsis field and click **Finished**. The Designer will return to the Function dialog and fill in the remaining popup and field.



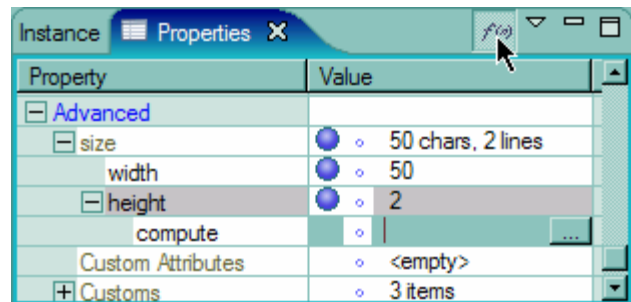
- j. Click **OK** to go back to the main function dialog and complete the formula:
- k. Select **== (equal to)** from the center popup and type “1” in the last field of the “If” section.
- l. Select **Choose a function** in the popup of the **Then** section and click the **Function** button.
- m. Choose **set** from the function popup.
- n. As the reference parameter, select **Choose an item on the form** from the popup, click the hand button and select the Synopsis field. Select “custom:my\_height” from the popup.

- o. As the value, select **Enter a number or text** from the popup and enter “viewer.measureHeight('chars', ‘’, ‘never’)”.
- p. Press **OK** to exit the Source Code Being Generated dialog.
- q. Press **Finish** to exit the Compute Wizard dialog.
- r. Within the custom option press the “...” button to bring up the compute wizard again; your formula should look like the following if it doesn't please fix it now (you must remove the single quotation marks surrounding the viewer.measureHeight function):

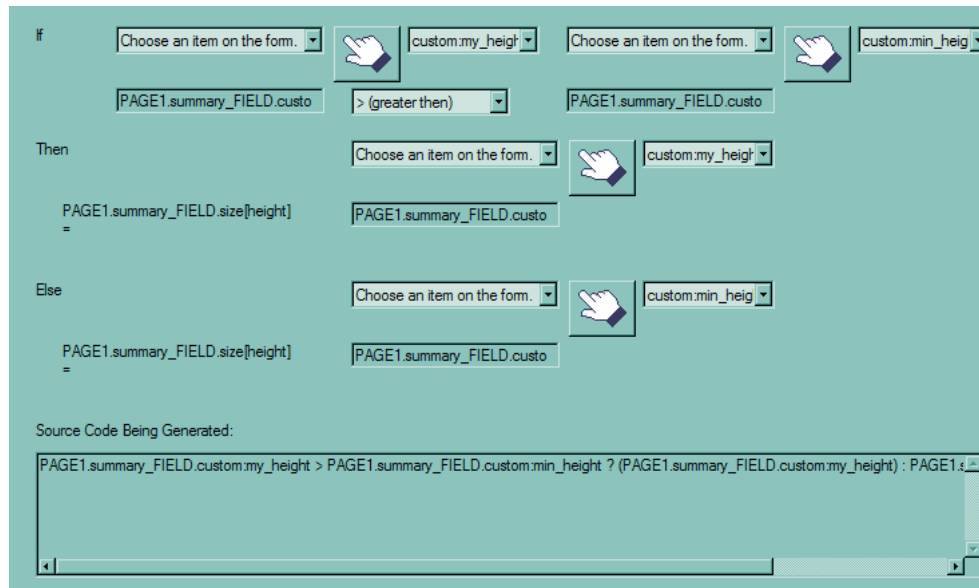
```
toggle(PAGE1.summary_FIELD.value) == '1'
? (set(' PAGE1.summary_FIELD.custom:my_height',
      "viewer.measureHeight('chars', '', 'never')"))
: ''
```

5. The next step is to add a compute that will make the height of the Synopsis field dependent on the variables “my\_height” and “min\_height”. If “my\_height” is greater than “min\_height” we want “my\_height” to be used as the height of the field, otherwise “min\_height” will be used. Remember that we set the default value of “min\_height” to '2'.

- a. Select the Synopsis field, in the Properties View expand the Advanced category and then expand the size option. Highlight the value of the height attribute and press the “**add the compute attribute**” button.



- b. Create a compute that sets the y coordinate of the field's size option to equal either the minimum height or actual height values.
- c. In English, if the height of the text is greater than the height of the field then make the field height equal to the height of the text, otherwise set the field height to the minimum, which is 2.
- d. Highlight the value of the compute attribute and press the “...” button. Create a compute that is **determined by a decision** and click **Next**.



- a. In the **If** section, **Choose an item on the form** and select the Synopsis field, then select **custom:my\_height** from the following popup. Select **> (greater than)** from the middle popup and then **choose an item on the form** and select the Synopsis field, then select **custom:min\_height** from the following popup
- b. In the **Then** section, **Choose an item on the form** and select the Synopsis field, then select **custom:my\_height** from the following popup. In the **Else** section, **Choose an item on the form** and select the Synopsis field, then select **custom:min\_height** from the following popup.
- c. The compute should look like the following:

```
PAGE1.summary_FIELD.custom:my_height >
PAGE1.summary_FIELD.custom:min_height ?
(PAGE1.summary_FIELD.custom:my_height) :
PAGE1.summary_FIELD.custom:min_height
```

6. Preview the form.
7. Enter several lines of text into the Synopsis field (more than 5), tab out of the field and watch the field grow!
8. Now to complete the third part of the exercise that will only show the scrollbars for the Synopsis field when it has the “focus” (the cursor is inside of the field).
  - a. Create a compute that disables the scrollbar when the focus leaves the field.
  - b. In the Properties View, select the **Scrollvert** option and press the **add a compute attribute** button.
  - c. When the field is “focused” the **scrollvert** option should contain “always”, otherwise “never”.

- d. Press the “...” button in the compute attribute and create a formula that is **determined by a decision** and click **Next**. Select **Choose an item on the form** and select **focused**, set the center popup to **equal to** and place the text “on” in the last field of the If section.
  - e. In the **then** field enter “always” and in the **else** field enter “never”.
  - f. Your compute should look like the following:  

```
PAGE1.summary_FIELD.focused == 'on' ? ('always') : 'never'
```
9. Preview your form again and this time you will notice that the scrollbar only appears when the cursor is in the Synopsis field.

## 4.10. Item Duplication

If a form contains a 'table' section, you may want to allow the user to dynamically add rows to the table. As with pages, this functionality requires the form to include a template. In this case, the template must be a row of items.

Usually, you place the template row on the same page as the table. It can be either the first row of the table, or be invisible so that it is only used for duplication purposes. Both techniques require about the same amount of work. If you use the first row of the table as the template row, you must clear the value options of all the newly duplicated items so that they do not contain the user input from the first row. If you hide the template row, the value options of the duplicated items will be blank, but you must remember to set their visible options to on.

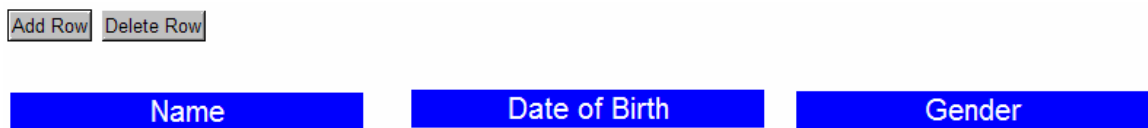
Setting up a template row of items requires a lot of configuration. You must configure the template row using relative positioning and relative item tags, so that duplicated items don't overlap the original items. Relative item tags include *itemnext* and *itemprevious*. Once you have configured each template item so that it is placed below or after the previous item in the build order, any new rows are automatically displayed in the correct position. Additionally, if you place any items below this table section, you must position them using relative positioning so that they move down as the user adds new rows.

As in duplicating pages, it is usually best to use an Add Row button to provide the user with a way to add additional rows. If you place this button below the template row within the form's build order, you can use it as a static item in the build order, allowing you to place newly duplicated items before it in the build order.

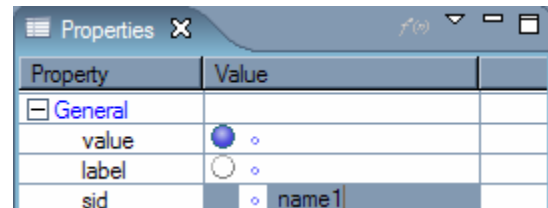
Other ways of triggering row duplication include adding a row every time the user leaves the last item of the last row. This method guarantees that there will always be an empty row.

## 4.11. XFDL Exercise 10 –Duplicating Rows of Items

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. “ItemDuplication”), then click **Finish**
  - g. The Designer should automatically switch to the Designer Perspective and open the file.
3. Create the first page of the form as shown below. Create two buttons, one to add a row and the second one to delete the last row. Create three labels with values set to ‘Name’, ‘Date of Birth’ and ‘Gender’.



4. Now we will create the first row which will be subsequently duplicated when the Add Row button is pressed.
  - a. Add a field to the canvas and place it below the Name label.
  - b. In the Properties View change its sid to “name1”.
5. The build order is very important in this exercise. We must insure that all the objects are created in the proper order for our relative alignment to work properly. We are going to position the name field relatively with the item that precedes it in the build order.
  - a. Right click on the name1 field and select **Relative Align > Relatively Align Below Previous Item**. You will notice that the name field is now sitting directly beneath the Gender label.



6. Now we want to align the name field with the Name label.
  - a. Left click on name1, hold the control key and left click on the name label to select both the items.
  - b. Right click on one of the two selected items and click on **Absolute Align > Absolutely Align Left to Left**.



Add Row Delete Row

Name	Date of Birth
<input type="text"/>	

7. Create another field with a sid of "dob1" and a popup with a sid of "gender1" and label of "Please Select".
8. Add two cells 'Male' and 'Female' in the **gender** popup.
9. Right click on **dob1** and select **Relatively Align > After Previous Item**.
10. Left click on dob1, hold the control key and left click on the date of birth label to select both the items. Right click on the label with both items still selected and click on **Absolute Align > Absolutely Align Left to Left**.
11. Right click on **gender1** and select **Relatively Align > After Previous Item**.
12. Left click on **dob1**, hold the control key and left click on the **gender** label to select both the items. Right click on the label with both items still selected and click on **Absolute Align > Absolutely Align Left to Left**.
13. Create a horizontal line with sid 'Line' below the fields.
  - a. The line should span from the **name** label to the **gender** label.
14. Select the **line** and select **Relatively Align > Below Previous Item**.



15. Left click on the **line**, hold the control key and left click on the name label to select both the items. Right click on the label with both items still selected and click on **Absolute Align > Absolutely Align Left to Left**.
16. In the Outline View move the line so it is located under gender1 in the build order.

Buttons: Add Row, Delete Row

Name	Date of Birth	Gender
<input type="text"/>	<input type="text"/>	Please Select ▼

17. We need a custom option variable to store the current row count.
  - a. Right click on the form and select **Wizards > Compute wizard**.
  - b. Press the **Create a Custom Option** button and create a new custom option in the custom namespace called **row\_count**.
  - c. Click **OK** and then **Cancel**.
  - d. Open the **Source View** and modify the value of the custom option so that it contains the number **1**.

Items Selected: PAGE1.global

Property to Set:

Create a Custom Option

Choose Namespace: custom

Option Name: row\_count

18. Now we need to implement the add row button.
  - a. Right click on the button and select **Wizards > Compute Wizard**.
  - b. Create a new custom option called **onclick**.
  - c. Click **OK** to get back to the main compute dialog.

Create a Custom Option

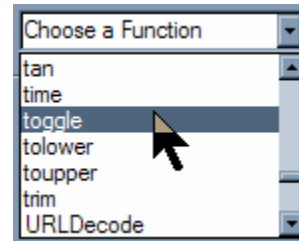
Choose Namespace: custom

Option Name: onclick

OK Cancel

- d. Now we add a toggle using the if/then/else decision.
  - i. Select **'The value is determined by a decision (If/Then/Else)'**
  - ii. Click **Next >**.

- e. In the If section of the compute, select **Choose a Function** from the popup.



- i. Click on the **Function** button and choose toggle in the next screen.

- f. Select **'Choose an item on the form'** in the reference.
  - i. Click the Add button using the hand tool and choose 'activated' from the drop down list.
  - ii. Choose **'Enter a number or text'** in the start and end drop downs.
  - iii. Enter 'off' in the start field and 'on' in the end field.
  - iv. Press **OK**.

- g. In the **Then** case, choose **'Choose a Function'** from the drop down list.
  - i. Click on the Function button and choose **'duplicate'** from the drop down list.
  - ii. Choose **'Enter a number or text'** from the reference drop down and enter **'name1'** in the field.
  - iii. Choose item from the type drop down list.
  - iv. Choose **'Enter a number or text'** from the newReference drop down and enter **'Line'** in the field.
  - v. Choose item from the newtype drop down list.

- vi. Choose **before\_sibling** from the location drop down list.
- vii. Choose **'Enter a number or text'** from the newName drop down and enter **'name'** in the field.
- viii. Press **OK** and then press **Finish** to exit the compute wizard.

- h. Go back into the wizard to edit the formula for **custom:onclick** manually. We need to duplicate the other items and also increase the row count by one each time Add Row is pressed. You may also change the toggle in the code view.
- i. Change the formula from

```
toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1'
? (duplicate('name1', 'item', 'Line', 'item',
            'before_sibling', 'name'))
: ''
```

to

```
toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1'
? (set('global.custom:row_count', global.custom:row_count + '1')
+. duplicate('name1', 'item', 'Line', 'item',
            'before_sibling', 'name' +. global.custom:row_count)
+. duplicate('dob1', 'item', 'Line', 'item',
            'before_sibling', 'dob' +. global.custom:row_count)
```

```

+. duplicate('gender1', 'item', 'Line', 'item',
            'before_sibling',
            'gender' + . global.custom:row_count) )
: ''

```

- j. Now preview the form and try pressing the Add Row button. To test the code that we just added, enter some data into the fields and then press the Add Row button.

Add Row Delete Row

Name	Date of Birth	Gender
<input type="text"/>	<input type="text"/>	Please Select ▼
<input type="text"/>	<input type="text"/>	Please Select ▼
<input type="text"/>	<input type="text"/>	Please Select ▼
<input type="text"/>	<input type="text"/>	Please Select ▼
<input type="text"/>	<input type="text"/>	Please Select ▼
<input type="text"/>	<input type="text"/>	Female Male

---

Note: In the example above, the first row is copied to create subsequent rows and hence any data entered in the first row will be copied over into the new cells. To get around this, you can delete the contents of a field after it is created or create invisible fields on the form to copy from. Once the invisible fields are duplicated, the `<visible>` option can be set to true.

---

- k. To clear the duplicated fields after they are created add appropriate set functions in the toggle. We will set the value of the newly created fields to ". You may change the toggle in the code view.

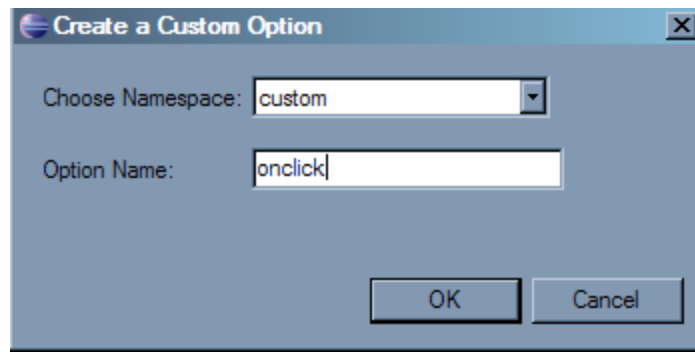
```

toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1'
? (set('global.custom:row_count', global.custom:row_count + '1')
+. duplicate('name1', 'item', 'Line', 'item', 'before_sibling',
            'name' + . global.custom:row_count)
+. duplicate('dob1', 'item', 'Line', 'item',
            'before_sibling', 'dob' + . global.custom:row_count)
+. duplicate('gender1', 'item', 'Line', 'item',
            'before_sibling',
            'gender' + . global.custom:row_count)
+. set('name' + . global.custom:row_count + '.value', '')
+. set('dob' + . global.custom:row_count + '.value', '')
+. set('gender' + . global.custom:row_count + '.value', '') )
: ''

```

### Deleting the Last Row

19. We must now use the destroy function just like we used the duplicate function and then decrement the `global.custom:row_count` by one each time a row is deleted.
20. Right click the **Delete Row** button and open the **Compute Wizard**. Create a new custom option called **onclick**.

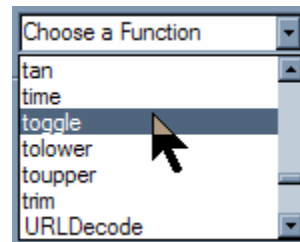


21. Now we add a toggle using the if/then/else decision.

- a. Select **'The value is determined by a decision (If/Then/Else)'** and click **Next >**.

22. Choose **'Function'** in the **If**.

- a. Click on the **'Function'** button and choose **toggle** in the next screen.



23. Select **'Choose an item on the form'** in the reference.

- a. Click the **Delete Row** button using the hand tool and choose 'activated' from the drop down list.
  - i. Choose **'Enter a number or text'** in the start and end drop downs.
  - ii. Enter **'off'** in the start field and **'on'** in the end field.
  - iii. Press **OK**.

24. In the **Then** case, choose '**Choose a Function**' from the drop down list.
  - a. Click on the Function button and choose '**destroy**' from the drop down list.
  - b. Choose '**Enter a number or text**' from the reference drop down and enter '**name**' in the field.
  - c. Choose **item** from the type drop down list.
  - d. Click **OK** to exit the Then case.
  - e. Click **Finish** to exit the compute wizard.

25. Go back into the wizard to edit the formula for **custom:onclick** manually. We need to duplicate the other items and also increase the row count by one each time Add Row is pressed.
26. Change the formula in the code view from

```
toggle(PAGE1.BUTTON2.activated, 'off', 'on') == '1'
? (destroy('name', 'item'))
: ''
```

to

```
toggle(PAGE1.BUTTON2.activated, 'off', 'on') == '1'  
? (destroy('name' + . global.custom:row_count, 'item')  
+. destroy('dob' + . global.custom:row_count, 'item')  
+. destroy('gender' + . global.custom:row_count, 'item')  
+. set('global.custom:row_count', global.custom:row_count - '1'))  
: ''
```

27. Now preview the form and try pressing the Add Row button. To test the code that we just added, add some rows to the form and press the Delete Row button.
28. The last step is insuring that the user is not able to delete the first row, since it is being used as the reference for adding new rows. We must set the active of the Delete Button to off if the row count is equal to 1.
29. Select the **Delete Button**, right-click and open the **compute wizard**.
30. Select active from the property popup, select **The value is determined by a decision** and click **Next**.
31. In the If section select **Choose an item on the form**, click the hand button and click somewhere on canvas (this selects the page global).
32. In the middle popup find **custom:row\_count** and then enter the number 1 in the last field of the If section.
33. In the Then section enter **off** and **on** in the else section.
34. Click **Finish**.
35. Preview the form, the Delete button should be inactive until a new row is added.

## 4.12. XFDL Exercise 11 – Creating Mutually Exclusive Checkboxes

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. “Checkboxes”), then click **Finish**
  - c. The Designer should automatically switch to the Designer Perspective and open the file. Create a label ‘Mutually Exclusive Checkboxes’ with a font size of 12 pts.

**abc** 3. Create a **label** with the value ‘**Please select your favorite Italian dish:**’.

4. Create four **checkboxes** under the label. Label the Checkboxes ‘Pizza’, ‘Lasagna’, ‘Spaghetti’ and ‘Calazone’. The final form should look as follows in the design view.

5. To make the checkboxes mutually exclusive so that the users can turn on only one checkbox at a time, we have to use the *toggle* and the *set* functions. The basic idea being that we *toggle* on the value of the checkboxes and when the value of a checkbox changes from ‘off’ to ‘on’, we *set* the value of the other checkboxes to ‘off’.
6. Let us create the toggle for the first checkbox. Right click on Pizza checkbox and click on **Wizards > Compute Wizard**.
7. Click on **Create Custom Option**, call the new option **valueOff**.
8. Click **OK**.
9. Check ‘the value is determined by a decision(If/Then/Else)’ and click Next.




10. Select '**Choose a function**' from the **If** popup list and click on the **Function** button.
11. Select **toggle** from the function popup list.
  - a. Choose the Pizza checkbox using the hand tool for the **reference**.
  - b. Select **Enter a number or text** from both the **start** and **end** drop down list.
  - c. Enter 'off' as the **start** value and 'off' as the **end** value.

Function Call

toggle

Parameters

reference: Choose an item on the form.  value PAGE1.CHECK1.value

start (optional): Enter a number or text. off


end (optional): Enter a number or text. on

- d. Click **OK** to exit the toggle function and return to if/then/else window.
- e. Type **1** in the **== (equal to)** field.
12. When the Pizza checkbox is switched from off to on, we want to set the value of the remaining checkboxes to off using the *set* function. Select **Choose a function** in the **Then** popup list and click on **Function**.
  - a. Select **set** from the function popup list. In the **reference** field, select the 'Lasagna' checkbox.
  - b. Select **Enter a number or text** in the **value** popup list and type *off* in the field.
  - c. Value is the new value that will be assigned to CHECK2.value. You can leave the **Type** and **Scheme** options blank.

Function Call

set

Parameters

reference: Choose an item on the form.  value PAGE1.CHECK2.value

value: Enter a number or text. off

type (optional): Choose a Source Type

scheme (optional): Choose a Source Type

13. Click **OK** to exit the set function.
14. Then click **Finish** to exit the if/then/else window. We still need to set the values of the last two checkboxes to **off**. We can do that in the code view.
15. Left click on Pizza checkbox and click on the **Source** tab at the bottom of the page. The start of the code relevant to the pizza checkbox should be highlighted in the source code view. Now change the toggle

**from**

```
<check sid="CHECK1">
  <itemlocation>
    <x>214</x>
    <y>185</y>
  </itemlocation>
  <custom:valueOff xfdl:compute="          &#xA;
    toggle(PAGE1.CHECK1.value, 'off', 'on') == '1' &#xA;
    ? set('PAGE1.CHECK2.value', 'off')          &#xA;
    : ''></custom:valueOff>
</check>
```

**to**

```
<check sid="CHECK1">
  <itemlocation>
    <x>234</x>
    <y>201</y>
  </itemlocation>
  <custom:valueOff xfdl:compute="          &#xA;
    toggle(PAGE1.CHECK1.value, 'off', 'on') == '1' &#xA;
    ? set('PAGE1.CHECK2.value', 'off')          &#xA;
    +. set('PAGE1.CHECK3.value', 'off')          &#xA;
    +. set('PAGE1.CHECK4.value', 'off')          &#xA;
    : ''></custom:valueOff>
</check>
```

---

Notice the use of +. to concatenate multiple statements and &#xA; to break up a line into multiple lines for readability purposes.

---

16. You may test the partially implemented form at this stage.
  - a. Preview the form and click on the Calazone checkbox.
  - b. Now if you click on the Pizza checkbox the Calazone checkbox should automatically be unselected.
17. Follow steps 7 through 15 for the other three checkboxes. As a shortcut you may create a custom option in each of the other checkboxes and then copy and paste the entire

compute. Then change the references within the set() statements. For example the complete code for the Lasagna checkbox should look like:

```
<check sid="CHECK2">
  <itemlocation>
    <x>102</x>
    <y>134</y>
    <align2t>CHECK1</align2t>
    <offsetx>193</offsetx>
  </itemlocation>
  <custom:valueOff2 xfdl:compute="          &#xA;
    toggle (PAGE1.CHECK2.value, 'off', 'on') == '1' &#xA;
    ? (set ('PAGE1.CHECK1.value', 'off')          &#xA;
    +. set ('PAGE1.CHECK3.value', 'off')          &#xA;
    +. set ('PAGE1.CHECK4.value', 'off'))        &#xA;
    : ''"></custom:valueOff2>
  <value>off</value>
  <label>Lasagna</label>
</check>
```

### Mutually Exclusive Checkboxes

Please select your favorite Italian dish:

Pizza	Lasagna	Spaghetti	Calazone
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

#### Bonus Exercise

Use custom options and the *getReference* function instead of *toggle* and *set* to make the checkboxes mutually exclusive. This approach is quite generic.

1. First, we need to create a custom option in the Pizza checkbox called `last_selected`. This option will hold the *sid* of the currently checked checkbox.
  - a. To do this right click on the pizza checkbox and select **Wizards > Compute Wizard**.
  - b. Click on **Create Custom Option** button.
  - c. Type **last\_selected** as the Option Name.
  - d. Click **OK** to return back to the compute wizard and then click **Cancel** to exit the compute wizard.
2. Now we will give `custom:last_select` a default value.

- a. Left click on the Pizza checkbox.
  - b. Select the Source view tab.
  - c. Add the sid of the Pizza checkbox in between the custom:last\_selected tags.  

```
<custom:last_selected>PAGE1.CHECK1</custom:last_selected>
```
  - d. Go back to the design view.
3. Each checkbox will also contain a custom option that holds their respective sid's.
- a. Right click on the Pizza checkbox and select **Wizards > Compute Wizard**.
  - b. Click on **Create Custom Option** button.
  - c. Type **my\_id** as the Option Name.
  - d. Click **OK** to return back to the compute wizard and then click **Cancel** to exit the compute wizard.
  - e. Select the **Source View** tab with the Pizza checkbox selected.
  - f. Create a compute inside the custom:my\_id option.
  - g. 

```
<custom:my_id xfdl:compute="getReference(", ", 'item')"></custom:my_id>
```
  - h. Save the form and go back to the design view.
4. Now we need to add a custom compute to the Pizza checkbox which will turn the last selected checkbox to off and set the value of custom:last\_selected to the sid of the Pizza checkbox.
5. To do so, right click on the Pizza checkbox and select **Wizards -> Compute Wizard**.
- a. Click on **Create Custom Option** button.
  - b. Type **my\_id** as the Option Name.
  - c. Click **OK** to return back to the compute wizard. Select **This decision is determined by a decision (If/Then/Else)**.
  - d. Click **Next** to go to (If/Then/Else) screen.
  - e. Select Choose a function from the **If** drop down list and click the **Function** button.
  - f. In the Function window,
    - i. Choose **toggle** from the drop down list.

- ii. Select **Choose an item on the form** from the reference list and use the hand tool to select the Pizza checkbox. Make sure that **value** is selected in the list next to the hand tool.
- iii. Select **Enter a number of text** in the start drop down list and the end drop down list.
- iv. Enter **off** in the start field and **on** in the end field.

The screenshot shows the 'Function Call' dialog box. At the top, the function name is 'toggle'. Under the 'Parameters' section, the 'reference' is set to 'Choose an item on the fom.' with a hand icon pointing to it. The 'value' dropdown is set to 'value', and the corresponding text field contains 'PAGE1.CHECK1.value'. The 'start (optional)' dropdown is 'Enter a number or text.' with a text field containing 'off'. The 'end (optional)' dropdown is also 'Enter a number or text.' with a text field containing 'on'.

- v. Click **OK** to return to the If/Then/Else window.
- g. Enter **1** in the field next to **== (equal to)** field.
  - h. We want to use the **set** function to set the last\_select value to off. To do so, select choose a function from the then drop down list and click on the **Function** button.
    - i. Choose **set** from the drop down list.
    - ii. Use the hand tool to select the Pizza checkbox.
    - iii. Choose custom:last\_selected from the drop down list next to the hand tool.
    - iv. Select **Enter a number or text** in the value drop down list.
    - v. Enter **off** in the value field. You may leave the type and the scheme options blank.

The screenshot shows the 'Function Call' dialog box. At the top, the function name is 'set'. Under the 'Parameters' section, the 'reference' is set to 'Choose an item on the fom.' with a hand icon pointing to it. The 'value' dropdown is set to 'custom:last\_selected', and the corresponding text field contains 'PAGE1.CHECK1.custom:last\_selected'. The 'value' dropdown is set to 'Enter a number or text.' with a text field containing 'off'. The 'type (optional)' dropdown is 'Choose a Source Type' with an empty text field. The 'scheme (optional)' dropdown is also 'Choose a Source Type' with an empty text field.

- vi. Click **OK** to get back to the If/Then/Else window.
- i. Click **Finish** to exit the compute wizard.
- j. Now we will finish the rest of the compute in the Source view. Select the Pizza checkbox and left click on the Source tab.
- k. We only want to change the **last\_selected** if the current checkbox was not the one in **last\_selected**.
  - i. Note that **custom:my\_id** contains the sid of the current checkbox.
  - ii. Remove the single quotes from **PAGE1.CHECK1.custom:last\_selected**.
  - iii. Also note that **last\_selected** contains the sid of the checkbox that needs to be turned off, hence we need to set the **custom:last\_selected.value** to off instead of just setting **custom:last\_selected** to off.

```
<custom:radio_behaviour xfdl:compute="
  toggle(PAGE1.CHECK1.value, 'off', 'on') == '1' and
    (CHECK1.custom:lastSelected != custom:my_id)
  ? (set(PAGE1.CHECK1.custom:last_selected +. \. Value', off))
  : ''"></custom:radio_behaviour>
```

6. Finally, to complete the toggle, we need to set the sid of the current checkbox to custom:lastselected. Change the toggle as follows

```
<custom:radio_behaviour xfdl:compute="
  toggle(PAGE1.CHECK1.value, 'off', 'on') == '1' and
    (CHECK1.custom:lastSelected != custom:my_id)
  ? (set(PAGE1.CHECK1.custom:last_selected +. \. Value', off)
    +. set('CHECK1.custom:last_selected', custom:my_id))
  : ''"></custom:radio_behaviour>
```

7. Now copy the **custom:radio\_behaviour** code to all the checkboxes to complete the exercise.

### Mutually Exclusive Checkboxes

Please select your favorite Italian dish:

Pizza	Lasagna	Spaghetti	Calazone
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## 4.13. Creating Signature Filters

Occasionally, you may find that you don't want certain items or options to be signed. You may wish to create different form sections to be signed by different users, or you may want a custom item to continue working after the form is signed. You can accomplish this with *signature filters*.

Signature filters allow you to set which form elements you want to sign. There are two kinds of signature filters:

- **omit** – Omit filters let you specify the form elements you don't want to sign, and ensures that the rest of the form is signed. This filter guarantees that *everything* in the form is secured, *except* the form elements that you choose.
- **keep** – Keep filters let you specify the form elements you want to sign, leaving the rest of the form unsigned. This filter only secures the form elements that you choose, leaving the remainder of the form unprotected.

---

**Note:** Omit filters provide greater security than keep filters. It is good practice to use omit filters rather than keep filters. If you must use a keep filter, use it in conjunction with an omit filter. For example, you might want to omit the items in a "For office use only" section from a user's signature, but you would secure the location and appearance of these items with a keep filter.

---

Workplace Forms Designer makes it easy to create secure signature filters. It allows you to select specific items that you want to omit from the signature, while automatically retaining item position information. This filter option is called *signitemrefs*. *Signitemrefs* allows you to specify individual items that the signature will omit or keep. For example, you might set the filter to omit BUTTON1 on PAGE1.

The Designer also provides an interface that assists you in creating custom signature filters. Custom signature filters allow you to omit or keep specific form elements, such as:

- **Items** – Specifies the types of items that the signature will omit or keep. For example, you might set the filter to omit all *button* items from the signature.
- **Options** – Specifies the types of options that the signature will omit or keep. For example, you might set the filter to omit all *triggeritem* options from the signature.
- **Groups** – Specifies one or more groups, as defined by the *group* option, that the signature will either omit or keep. This filters any radio buttons or cells belonging to that group, but does not filter *list*, *popup*, or *combobox* items. For example, if you had a popup containing a cell for each State, you might set the filter to omit the *State* group, which would omit all cells in that group.
- **Data groups** – Specifies one or more datagroups, as defined by the *datagroup* option, that the signature will either omit or keep. This filters data items belonging to that datagroup, but does not filter any *action*, *button*, or *cell* items. For example, if you

had an enclosure button containing references, you might set a filter to omit the *References* datagroup, which would omit all data items in that group.

XFDL also allows you to filter item, option, and page references. While these options are not currently available for custom signatures while using the Designer, you can add these options in the Designer's Code View window:

- ***signoptionrefs*** – Specifies individual options that the signature will omit or keep. For example, you might set the filter to omit `BUTTON1.value` on `PAGE1`.
- ***signpagerefs*** – Specifies individual pages that the signature will omit or keep. For example, you might set the filter to omit `PAGE1`. The following code sample illustrates a *signoptionrefs* filter that omits a checkbox's *visible* option:

```
<signoptionrefs>  
<ae>omits</ae>  
<ae>PAGE1.accept_CHECKBOX.visible</ae>  
</signoptionrefs>
```

For more information on creating digital signatures and their filters, see “*Creating Signature Buttons in XFDL*” and the *XFDL Specification*. These documents are available on the IBM Workplace Forms Support web site (<http://www.ibm.com/software/support/workplaceforms>).



## 4.14. XFDL Exercise 12 – Creating Signature Filters

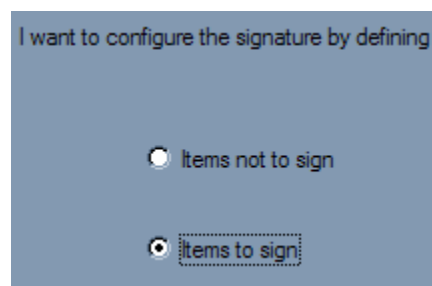
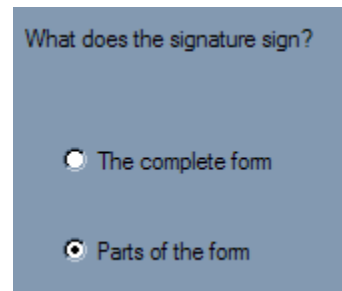
This exercise uses a Travel Authorization form that contains three sections. Currently, all three sections are mandatory and active at the same time. Modify this form so that the status of each section is dependent on the previous section being signed. In other words, when the form is first opened, Section I should be mandatory and active. When this section is complete and signed, Section II should become mandatory and active. Finally, when Section II is signed, Section III should become mandatory and active.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Switch to the Resource Perspective and open **XFDL\_Ex12\_SignatureFilters.xfd** from the **TrainingExercises\_XFDL** project in the Navigator View.

### Setting up Signature Filters

3. All the signature buttons have been configured such that their **Type** is set to **Signature**.
4. Since all the buttons only sign their respective sections, we need to omit other sections from each signature button.

- a. Right click on Traveler Signature button and click on **Wizards > Signature Wizards**.
- b. Click on **Parts of the form** radio button and click **Next**.
- c. Click on **Items to sign** and click **Next**.



- d. Page1 should be under the **Not Signed** column. Click **Next** to go to the next step.

- e. Use the hand tool to select all the items under the **Travel Request** section of the form. To select multiple fields, hold down the “Ctrl” key and left-click the mouse

**Travel Request**

I. Travel Info

Traveller Name:

Destination:

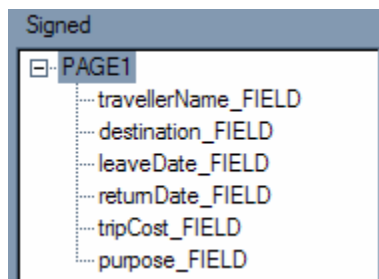
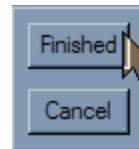
Leave Date:      Return Date:      Estimated Trip Cost:

Purpose of Travel:

Traveller's Signature:

button.

- f. Click **Finished** to return to the **Signature Wizard**.
- g. You should have all the items under the **Travel Info** section marked to be signed. Click **Next**.



- h. Choose your signature. We will choose **Clickwrap** for this example. Click **Next**.
- i. Enter a suitable Title for the Signature and a **Prompt** (Answer the following question?). Enter a **Safeguard Question** and provide a **Default Answer**.

Title:  Main Text:

Prompt:

Safeguard Question List:

	Question	Default Answer
<input type="button" value="DEL"/>	<input type="text" value="5 + 5 ="/>	<input type="text" value="10"/>

- j. Click **Finish** to exit the signature wizard.
5. Preview the form and test the signature. You should not be able to alter any items in the **Travel Info** section once the form is signed using the **Traveller's Signature** button.
  6. Select the Supervisor's signature and repeat **step 4**.
    - a. Once you have completed selecting all of the objects, press **Finished** to go back to the Filter dialog. Click **Next**.
    - b. Check to make sure the desired items are covered under the signature. Click **Next**.
    - c. Choose your signature. We will choose **Clickwrap** for this example. Click **Next**.
    - d. Enter a suitable Title for the Signature and a **Prompt** (Answer the following question?). Enter a **Safeguard Question** and provide a **Default Answer**

Title:  Main Text:

Prompt:

Safeguard Question List:

	Question	Default Answer
<input type="button" value="DEL"/>	<input type="text" value="What is 4 * 20 :"/>	<input type="text" value="80"/>

- e. Click **Finish** to exit the **Signature compute** wizard.
7. Finally we want the finance officer's signature to sign the entire form.
    - a. Right click on Finance Officer's Signature button and click on **Wizards > Signature Wizards**.
    - b. Click on **The complete form** radio button and click **Next**.
    - c. Choose your signature. We will choose **Clickwrap** for this example. Click **Next**.
    - d. Enter a suitable Title for the Signature and a **Prompt** (Answer the following question?). Enter a **Safeguard Question** and provide a **Default Answer**.

- e. Click **Finish** to exit the **Signature compute** wizard.

### Building Workflow by Setting Fields Inactive

Because the form simulates a workflow, the implied order is from Employee to Supervisor to Finance Officer. We should not be able to fill out the sections out of order and this is what we are going to enforce by using a few simple computes.

1. Right click the **Supervisor's Signature** button and select **Wizards > Compute Wizard**.
2. Change the property to **active** and click on **The value is determined by a decision (If/Then/Else)** radio button. Click **Next**.

Items Selected: PAGE1.SIG2\_BUTTON

Property to Set: active Create Custom Option

The value is equal to another form item.

The value is equal to a function.

The value is set by a calculation of two values.

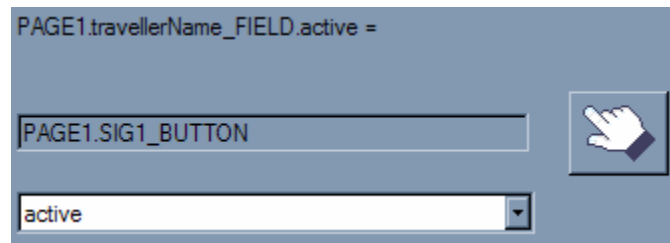
The value is equal to the sum of multiple fields on the form.

The value is determined by a decision (If/Then/Else).

The value is set by a manually created formula.

3. Now we need to complete the If/Then/Else block
  - a. In the first **If** popup select **Choose an item on the form** and use the hand tool to select the **Travelers Signature** button. Choose **Signer** in the drop down next to the hand tool.
  - b. Then in the middle popup select **== (is equal to)**.
  - c. The last popup in the **If** section should be **a number or text** and the field below it should be empty.
  - d. In the **Then** popup select **off**.
  - e. In the **Else** popup select **on**.
  - f. Press **Finish** to exit the Formula dialog.

4. Select the **Finance Officer's** signature and **repeat steps 1-3**. References to "SIG2\_BUTTON" should be changed to "SIG3\_BUTTON".
5. The next step is to Configure the Field and Labels' Active State under each section.
  - a. This step requires that we setup similar objects at the same time. So do the checkboxes together, the labels together and the fields together. Do one section at a time.
  - b. Select an object (or more than one of the same type), right click and select **Wizards > Compute Wizard**.
  - c. Set the **Property to Active** and select **The value is equal to another form item**. Click **Next**.
  - d. Use the hand tool to select the **Traveler Signature** button. Choose **Active** from the popup list below the hand tool.

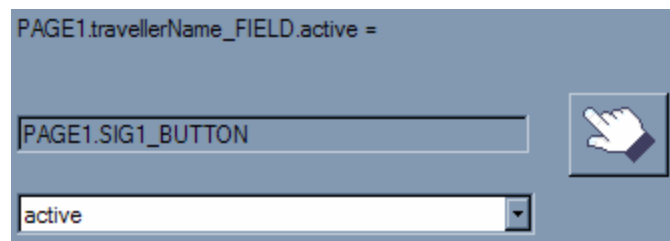


- e. Click **Finish** to exit the compute wizard and return to the form in the design view.
6. Repeat until all the fields of each section have been configured.

#### Configure the Field's Mandatory State

When the signature button of a section is active then we want the fields of that section to be mandatory.

1. Select an object, right click and select **Wizards > Compute Wizard**.
  - a. Set the Property to **Mandatory** and select **The value is equal to another form item**. Click **Next >**.
  - b. Use the hand tool to select the **Traveler Signature** button. Choose **Active** from the popup list below the hand tool.



- c. Click **Finish** to exit the compute wizard and return to the form in the design view.
2. Repeat until all the fields of each section has been configured.

## 4.15. XFDL Exercise 13 – Implementing Web Services in XFDL

This exercise demonstrates how to implement a Web Service within a form. This assumes that you already have a WSDL that can be used as the starting point.

### Prerequisites

- This exercise requires that the student have access to the Internet.
- This exercise is dependent on a Web Service that is not hosted by IBM and therefore its availability can not be guaranteed. The service used in this exercise can be substituted for any other valid Web Service.

### Exercise

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form, called XFDL\_Ex13\_Web Services.xfd, in the TrainingExercises\_XFDL project.
  - a. Select **File > New > New Workplace Form**
  - b. Select **TrainingExercises\_XFDL** as the parent folder
  - c. Enter **XFDL\_Ex13\_Web Services.xfd** as the file name
  - d. Click **Next**
  - e. Select **Default Empty Form – XFDL**
  - f. Click **Finish**
3. Enclose the Web Service WSDL called **XFDL\_Ex13\_spellcheck.wsdl** that is included in the **TrainingExercises\_XFDL** project.
  - a. Open the **Enclosures View** and expand **WSDL**.
  - b. Right-click on **Web Services** and select **Enclose WSDL File**.
  - c. In the new dialog that appears, browse to the location of the **XFDL\_Ex13\_spellcheck.wsdl** file and click **Open**.
  - d. In the **Enclosures View** if you expand **Web Services** you should now see that the WSDL has been added to the form.
4. Add a **label**, a **field** and a **button** to the Designer Canvas.
  - a. Position and layout are not crucial here. Feel free to “dress up” the form as you like.
  - b. The field will contain the word to spell check.

- i. Change the **label** property of the field to 'Enter a word to spell check'
  - c. The label will display the result of the Web Service.
    - i. Change the **value** property of the label to 'Web Service Result'
  - d. The button will trigger the call to the Web Service.
    - i. Change the **value** property of the button to 'Check Spelling'
5. When the button is pressed we want to call the Web Service and place the result into the label. (To accomplish this we will require a compute on the 'activated' state of the button).
  - a. Right-click the button and select **Wizards > Compute Wizard**.
  - b. We want to place this compute in a custom option.
    - i. Click the **Create Custom** Option button.
    - ii. The namespace should be custom and the option name will be **onClick**.
    - iii. Click **OK**.
  - c. The **Property to Set** should already contain the custom option that you just created called **onClick**; if it doesn't then select it from the popup.
  - d. Select **The value is determined by a decision** and click **Next >**.
  - e. In the If popup select **Choose a Function** and click the **Function** button.
    - i. Select **toggle** from the popup
    - ii. Select **Choose an item on the form** as the reference and click the button with the hand.
    - iii. Click the "Check Spelling" button and then click **Finished**.
    - iv. Select **activated** from the popup that follows the button with the hand.
    - v. Select **Enter a number or text** from the **start** popup and enter **off** as the value
    - vi. Select **Enter a number or text** from the **end** popup and enter **on** as the value
    - vii. Click **OK**
  - f. Make sure that **== (equal to)** is selected and then type **1** in the next field.
  - g. In the **Then** popup select **Choose a function** and click the **Function** button.



- i. Select **set** from the function popup.
  - ii. Select **Choose an item on the form** as the reference and click the button with the hand.
  - iii. Click on the label and then click **Finished**
  - iv. Select **value** from the popup that follows the button with the hand.
  - v. Select **Enter a number or text** as the value and then enter **GoogleSearchService\_GoogleSearchPort.doSpellingSuggestion('1tnBlt5QFHLMq2jrDsaTUtK40yw/s6Fy',PAGE1.FIELD1.value)** as the value.
  - vi. Click **OK**.
- h. Click **Finish**.
- i. The compute is pretty much complete but there is one change that must be made before it will work correctly. When we entered the Web Service function name as a number or text the Designer surrounded it in single quotes; we must remove these quotes.
    - i. Select the button, right-click and open the Compute Wizard again.
    - ii. Select the **custom:onClick** property from the **Property to Set** popup
    - iii. This time the compute can be seen in the field at the bottom of the dialog. Inside of that field remove the single quotes that surround the second parameter of the set function.
 

```
toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1' ?
  (set('PAGE1.LABEL1.value',
    'GoogleSearchService_GoogleSearchPort.doSpellingSuggestion('1tnBlt5QFHLMq2jrDsaTUtK40yw/s6Fy',PAGE1.FIELD1.value)')) : ''
```

becomes

```
toggle(PAGE1.BUTTON1.activated, 'off', 'on') == '1' ?
  (set('PAGE1.LABEL1.value',
    GoogleSearchService_GoogleSearchPort.doSpellingSuggestion('1tnBlt5QFHLMq2jrDsaTUtK40yw/s6Fy',PAGE1.FIELD1.value)
  )) : ''
```
    - iv. When you are done modifying the compute, click **Finish**. Click **OK**.
  - j. Now preview the form to test the Web Service. Keep in mind that you must have access to the internet and this Web Service must be active. **This service is not maintained by IBM and is therefore not guaranteed to work at all times.**

- i. In the field, type 'allowes'
- ii. Click the 'Check Spelling' button.
- iii. The label is updated to show the correct spelling, 'allowed'.

## 5. Building Simple XForms Forms

### 5.1. Audience

This section is intended to teach the reader how to create simple forms that implement XForms. Building simple XForms forms are primarily schema/instance driven; therefore the user should understand the basic structure of an XML instance. This module will enable you to create a static form that uses XForms UI Controls.

### 5.2. Overview

A new blank XForms form can be created using the 'Default Empty Form – XForms' template provided in the Workplace Forms Designer. If a different template is selected that does not have XForms support, the user will not have access to any XForms elements until XForms support has been enabled, and both a model and instance have been created.

### 5.3. Adding XForms Support – Template vs. Enable

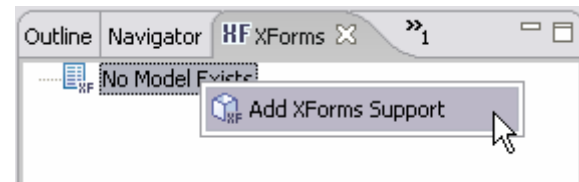
XForms support is required if your form will use XML instances, schemas, or XForms. There are two ways to Add XForms support to your XFDL forms:

1. While creating a new form

If you create your new form using the **Default Empty Form – XForms** template the Designer will automatically enable XForms.

2. Modifying an existing XFDL form

To enable XForms support for an existing form, open the **XForms View**, right click on the "No Model Exists" element and select "Add XForms Support".

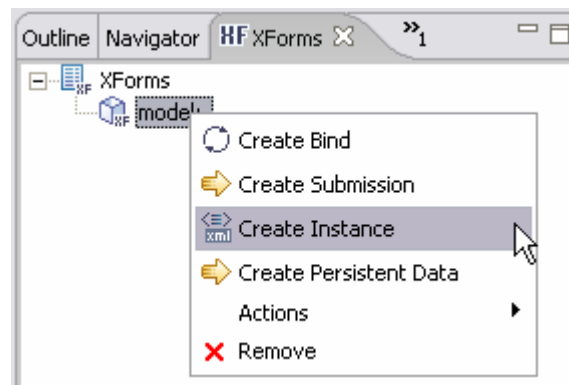


Adding XForms Support, with either method, will create a default model that can be used to enclose XForms Instances.

## 5.4. Creating XForms Instances – Various Methods

An XForms Instance is the mechanism that is used to embed arbitrary XML into a form. The instance data can be used to store input values, pre-populate fields with data, or dynamically generate list selections.

There are several different ways that an instance can be created. To create an instance from within the **XForms View**, right-click the **model element** and select **Create Instance**. Once the instance is created the user must define its structure.



Instances can also be created in the Instance View. There are three different mechanisms that can be used to create an Instance:

1. Creating a default instance that does not contain any children
2. Generating an instance based on the items in the current document
3. Creating a new instance from a web service
4. Creating an instance based on a schema file

Each procedure is described below in further detail. Once you are finished creating your instance, continue on to section 5.5: “Creating XForms Objects”.

### 5.4.1. Creating a new empty instance



Creating a new instance will create the default data instance:

```
<xforms:instance id="INSTANCE1" xmlns="">
  <data>
  </data>
</xforms:instance>
```

The user must then modify the instance manually. Several manual operations are provided; elements can be added, renamed, given default values, given attributes or removed.

#### Adding Elements

Elements can either be added to the xml instance as children or as a sibling.

To add a child element to the instance select the parent element, right-click then choose Add Element. A new element will be created beneath the selected element. If the parent already has children then the new child will be added after the existing elements. Siblings can only be

added before the selected element. Choose an item then right-click and select Insert Element Before; a new element will be created before the one that was selected.

### Changing an Element's Name

Double-click the element, in the Instance View, to change its name; a field will appear allowing the user to enter a name. Note: Element names must not contain special characters (i.e. spaces, <, >, &, etc).

### Adding a Default Value to an Element

If you want to give the element a default value, right-click and select Add Text. This will create a sub-element called "Text"; double-click this new element and enter the desired default value.

### Adding an Attribute to an Element

If you want to add an attribute to an element, right-click and select Add Attribute. This will create a "child" that can be renamed and given a value. Both of these operations can be found in the right-click menu; they are Rename and Edit, respectively.

### Removing Elements

Elements can be removed from the Instance View in two ways: pressing the delete button and selecting "Delete" from the right-click menu.

## 5.4.2. Creating an instance from the current form



If the form has already been created an instance can be generated that will mirror the structure of the current form. The generated instance of a form that has one page and two fields would appear as follows:

```
<xforms:instance id="INSTANCE" xmlns="">
  <document>
    <PAGE1>
      <FIELD1></FIELD1>
      <FIELD2></FIELD2>
    </PAGE1>
  </document>
</xforms:instance>
```

The operation uses the sid of each object as the element name. Every form item is represented in the resulting instance. This instance shows the page as a parent element and each item on the form page as a child element.

### 5.4.3. Creating a new instance from a Web Service



If a WSDL has been enclosed within a form then the Workplace Forms Designer can create an empty instance that can be used to store the request and response of a Web Service. To create an instance from an enclosed WSDL, open the Instance View and click on the “Create Instance from a WSDL” button.

### 5.4.4. Creating a new instance from a schema





























If a schema has been enclosed within a form then the Workplace Forms Designer can create an empty instance that adheres to its rules. To create an instance from an enclosed schema, open the Instance View and click on the “Create Instance from a Schema” button.

## 5.5. Creating XForms Objects

XForms is a method of abstracting the rules and format of the data collection process from the interface. XForms defines UI Controls that generically define what type of input is expected and stores the received data into an XML instance. XForms is less concerned about the presentation of the form, since its purpose is to provide business rules that can be implemented across multiple mediums; one of which is XFDL. XFDL “wraps” the XForms UI Controls providing finer control over the presentation layer of the form.

The XForms drawer and items will only be available if the form contains a valid XForms model. For more details on creating an XForms model please refer to section 5.3 “Adding XForms Support”. For every XFDL item there is a corresponding XForms item; an XForms item is comprised of the original XFDL code and a corresponding XForms expression. For complete item definitions or specific implementation details please refer to IBM’s XFDL 2.6 specification.

Icon	Item	Description
	Action (Submit)	An XFDL action that is linked to an XForms submission, and will submit the XForms data to a specified URL.
	Action (Trigger)	An XFDL action that executes one or more XForms actions when it is activated.
	Button (Submit)	A button that is linked to an XForms submission, and will submit the XForms data to a specified URL.
	Button (Trigger)	A button that executes one or more XForms actions when it is pressed.
	Button (Upload)	A button that allows the user to attach a file to the form.
	Case	A pane object that is used with the XForms Switch, only one case can be visible at a time.
	Check (Input)	A standard checkbox.
	CheckGroup (Select)	A control that allows check boxes to be grouped together and supports selecting multiple items.
	CheckGroup (Select1)	A control that allows check boxes to be grouped, but only one may be selected at a time.
	Choice (Item)	Allows user to add an item to a CheckGroup, RadioGroup, Popup, Combobox or List.
	Combobox (Select1)	A Combobox that only allows the user to select one item or type their own entry.

	Custom Item	An item that exists in the “custom” namespace, used to contain a variable or compute.
	Date Picker (Input)	A date field with a button that opens a calendar.
	Field (Input)	A standard one line field.
	Field (Secret)	A standard field where all input appears as an asterisk.
	Field (TextArea)	A multi-line field.
	Label (Output)	A standard label.
	List (Select)	A list box that supports the selection of multiple items.
	List (Select1)	A list that only supports selecting one item at a time.
	Pane (Group)	Creates a container that can be used to group multiple objects together.
	Pane (Switch)	The XForms control that allows multiple panes to overlap each other, only one pane is visible at a time.
	Popup (Select1)	A popup that only supports the selection of one item at a time.
	RadioGroup (Select1)	A control that allows radio buttons to be grouped, but only one may be selected at a time.
	Slider (Range)	A slider object that can be used to represent a range.
	Table (Repeat)	A repeating section (table) that is generated from XML data in the XForms instance.
	Table (Repeat) By Wizard	A repeating section (table) that uses a wizard to help the user create their table.

The process for creating an XForms item is the same as that for creating an XFDL item. Select the item you wish to create from the Design Palette and then click on the Design canvas.

In addition to the original XFDL properties, XForms items have properties that are specific to XForms. The XForms properties can be found in the Properties View under the XForms category.

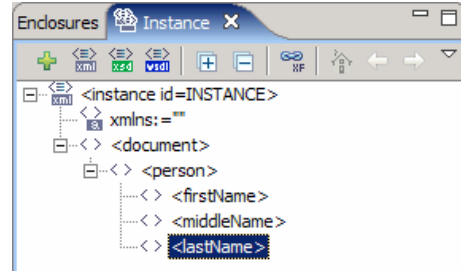
For additional information regarding the XForms items please refer to the XFDL specification.



## 5.6. Creating Items from an XForms Instance

Form items can be generated directly from an XML Instance. You can choose to create items one at a time or grouped by parent elements. If your form contains an XML Instance:

5. Open the **Instance View** and expand the instance so you can see all the elements. In this example the XML instance has a parent element that contains several children.
6. Select a single element or a parent element that contains all the items that you want to create. If you select the “lastName” element then a single field will be created and the contents will be stored in the instance. If you select the “person” element then three fields will be created (one for each child) grouped within a pane.
7. Click and hold the left-mouse button and drag the selected element onto the canvas. This will create a field for every child element contained within the selected element.



Note: Items created using drag and drop from the Instance View now get created with corresponding labels. The label sid is a concatenation of the field name, “\_LABEL” and a unique number (an increment starting with 1).

firstName	<input type="text"/>
middleName	<input type="text"/>
lastName	<input type="text"/>

## 5.7. Links to Data Model

One of the main differences between XFDL and XForms items is that the latter stores its data in an XML data instance. Every XForms item **must** be bound to an instance element. Every XForms item has a “ref” attribute that defines the instance element where its data should be stored. The “ref” attribute uses an XPath statement to reference the instance. Fortunately, the Designer will create simple versions of these statements for you, but as the complexity of your form increases you may be required to write a few of these on your own.

If your form already contains an instance and the corresponding fields, then the next step is to “bind” them together so that when data is entered into an item it is stored in the instance. Open the Instance View, expand the instance, and locate the first element. Select the element and drag it onto a form item to create a link between the two objects.

Let’s break down an example reference.

```
<xforms:label></xforms:label>
<xforms:value></xforms:value>
```

```
<instance id="personData">
  <data>
    <person>
      <name></name>
    </person>
  </data>
</instance>
```

The first part of the reference is pointing to the proper instance, instance(*theInstanceID*), in this case the instance id is personData. Once the instance has been defined, we traverse the elements naming each one until we get to the one that we want.

It is important to note that the first child element of an instance is the root node. It should not be included in XPath references to elements inside the instance. Therefore, the reference for the name element is **person/name** not **data/person/name**.

### 5.7.1. Highlighting Bound Items

The Designer provides a quick tool that allows you to see if a relationship has been established between the form objects and the XML instance data. This tool has three modes: highlight objects with binds, highlight object without binds and hide bind highlighting. Working with a complex busy form can be challenging and binds are an easy thing to forget. Being able to quickly determine which items have binds or which items do not will greatly decrease the time it takes to repair those mistakes. This tool is located in the toolbar at the top of the Forms Designer.



#### Highlight Objects With Binds

Activating the button highlights all of the input items that have an associated XForms bind.



### **Highlight Objects Missing Binds**

Activating the button a second time highlights all of the input items that do not have an associated XForms bind.



### **Hide Bind Highlighting**

Activating the button a third time will turn off the bind highlighting.

## 5.8. XForms Exercise 1 – Build Simple XForms Forms

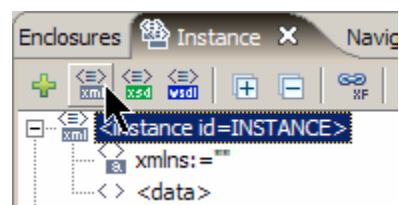
Now that we have discussed the basic XForms elements and how to bind them to instance data it is your turn! The purpose of this exercise is to give you an opportunity to explore the creation and modification of XForms items. We will be creating the exact same form as in the XFDL module. This time we will be using XForms items instead of the traditional XFDL items.

**XFORMS Items**

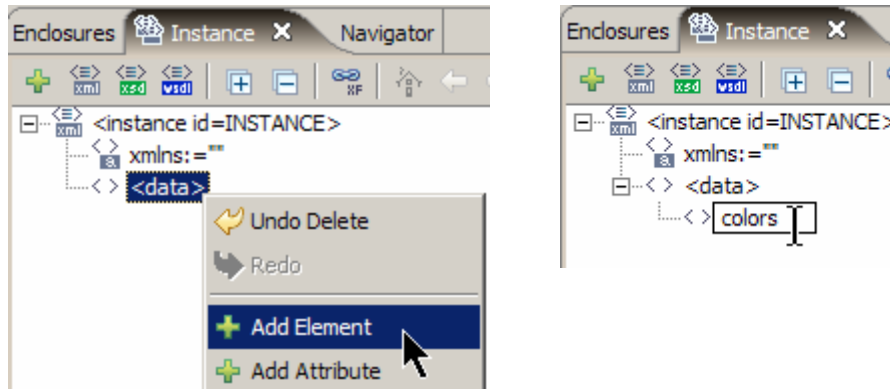
1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new project
  - a. **File > New > Project**
  - b. The Simple Project should be selected (select it if it isn't), then click **Next**
  - c. Enter a project name into the "Project Name" field (i.e. "FormExamples"), then click **Finish**
3. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. "XFormsPaneEx")
  - c. Select the **Default Empty Form – XForms** template, then click **Finish**
  - d. The Designer should automatically switch to the Designer Perspective and open the file.

### Add a popup item to the form.

1. First we need to create an xml instance and populate it with our data. To create an xml instance click on the xml button on the **Instance View** as shown here.

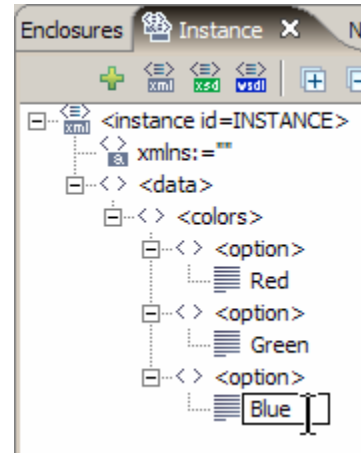
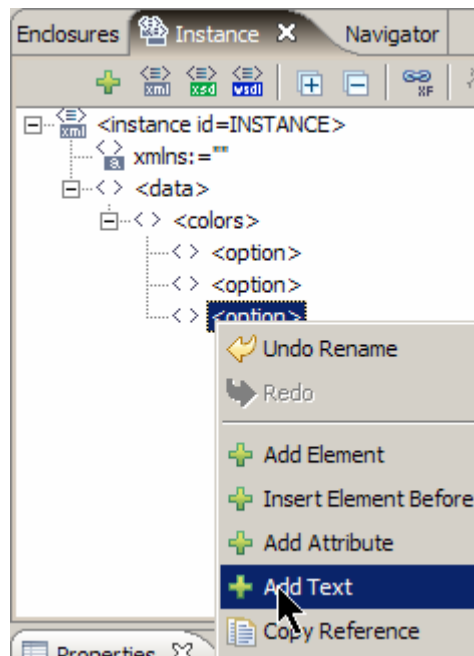
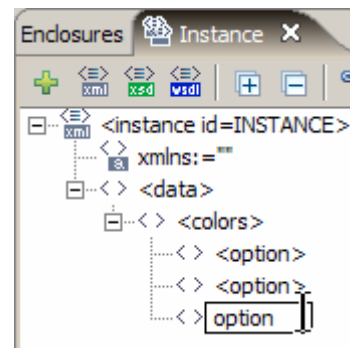


2. Now Add an element called 'colors' to the instance as a child of the data element.



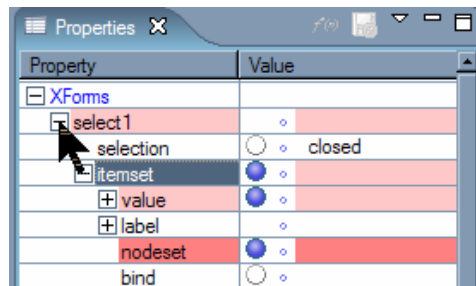
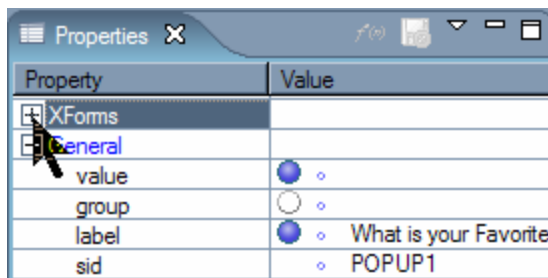
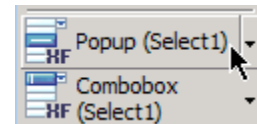
3. Next add 3 elements to the instance as the children of the colors element.

- Rename each of 3 child elements to "option".
- Add a value to each of the options.
  - Right click on the **option** element and choose "add text".
  - Add three option elements; **Red**, **Green** and **Blue**.



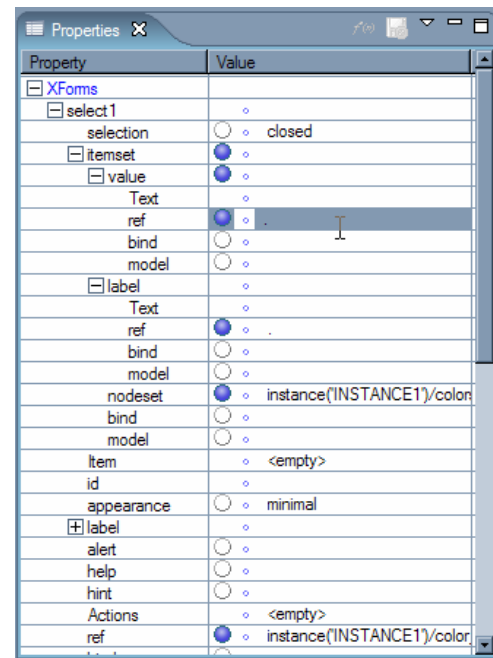
4. To create a popup from a data instance, use the **select1** control. The select1 control limits users to a single choice.

- Left click on **Popup (Select1)** from the Standard Library on the Palette.
- Left click on the canvas to place the popup select1 control on the form.
- Set the value of the **label** property to 'What is your Favorite Color?'
- Expand the XForms control in the properties of the popup item.



5. **Itemset** allows you to create lists and groups that offer dynamic choices that are determined while the form is in use. Itemset has one attribute called Nodeset, which is the path to the group of nodes that provide the list options.

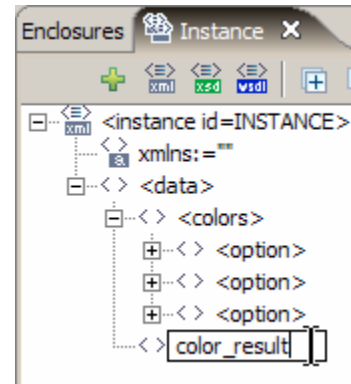
- The label option is a mandatory element of the itemset control. This label is used to provide the choices for the list. Using a reference to the attributes of the <option> nodes, label creates and displays the list of choices that are displayed to the user.
- The value option indicates which options are selected. Value has one attribute called ref, which is the path to the instance node.



- Set the **itemset > value > ref** and **itemset > label > ref** to '.' and the **nodeset** to **instance('INSTANCE')/colors/option**.
  - The simplest method of setting the **nodeset** is to open the **Instance View**
  - Select one of the **colors > option** elements, right-click and choose **Copy Reference**.

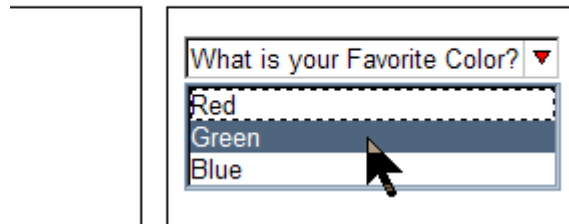
- iii. In the **Properties View**, find the **nodeset** property and paste the copied reference into its value.

- 6. Add another node to the instance called **color\_result** to store the result of the choice.



- 7. Set **Select1 > ref** to **instance('INSTANCE')/color\_result**.
  - a. To accomplish this task, open the **Instance View**
  - b. Select the **color\_result** element and drag it over top of the “color” popup. This will automatically set the **Select1 > ref** to the intended value.
- 8. Preview the form.

## XFORMS Items

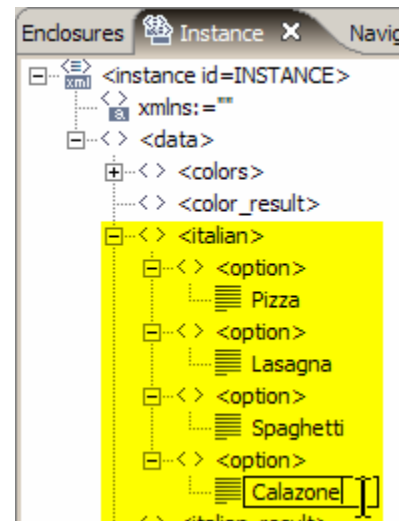


### Add a list item to the form

- 1. Create a child node called **Italian** to the xml instance created in step 1.
- 2. Create a child node under **Italian** called **option** for **Pizza, Lasagna, Spaghetti** and **Calazone**.
- 3. To create a list from a data instance, use the **select1** or **select** control. The **select1** control limits users to a single choice.



- a. Left-click on **List (Select1)** from the Standard Library on the Palette.

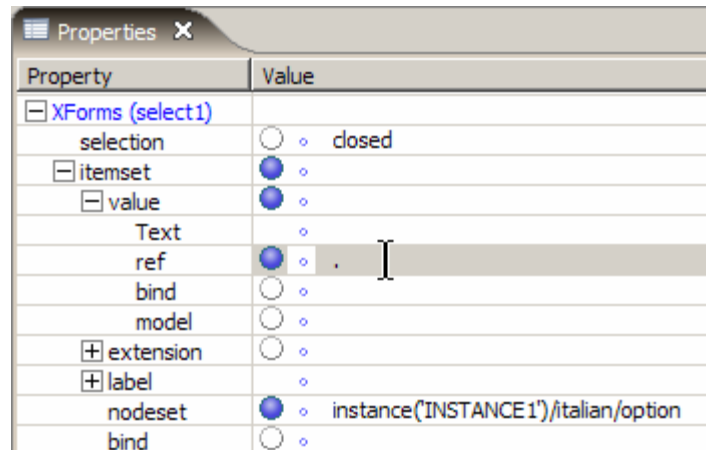


- b. Left-click on the canvas to place the list select1 control on the form.
4. Set the value of the **label** property to **'What is your favorite Italian dish?'**
5. Expand the XForms control in the properties of the popup item.
6. Itemset allows you to create lists and groups that offer dynamic choices that are determined while the form is in use. Itemset has one attribute called Nodeset, which is the path to the group of nodes that provide the list options.

a. The label option is a mandatory element of the itemset control. This label is used to provide the choices for the list. Using a reference to the attributes of the `<option>` nodes, label creates and displays the list of choices that are displayed to the user.

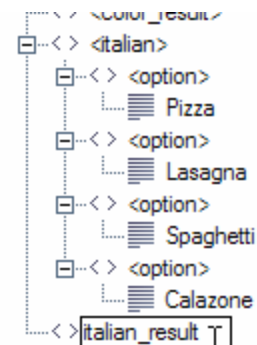
b. The value option indicates which options are selected. Value has one attribute called `ref`, which is the path to the instance node.

c. Set the **itemset > value > ref** and **itemset > label > ref** to `'.'` and the **nodeset** to **`instance('INSTANCE')/italian/option`**.

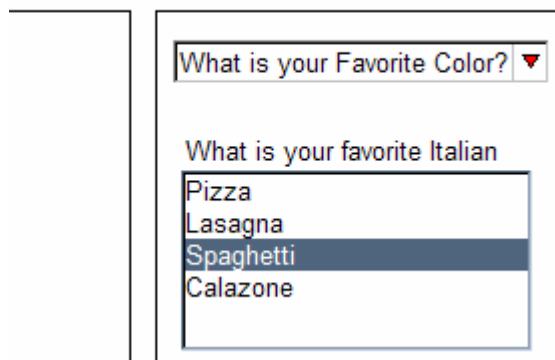


Property	Value
XForms (select1)	
selection	<input type="radio"/> closed
itemset	<input checked="" type="radio"/>
value	<input checked="" type="radio"/>
Text	<input type="radio"/>
ref	<input checked="" type="radio"/> .
bind	<input type="radio"/>
model	<input type="radio"/>
extension	<input type="radio"/>
label	<input type="radio"/>
nodeset	<input checked="" type="radio"/> instance('INSTANCE')/italian/option
bind	<input type="radio"/>

7. Add another node called **italian\_result** to the instance to store the result of the choice.
8. After creating the instance element, select the node in the **Instance View** and drag it onto the List object on the canvas. This will create a link between the form item and the instance data, that way when a selection is made it will be stored in the instance element.
9. Preview the form.



## XFORMS Items

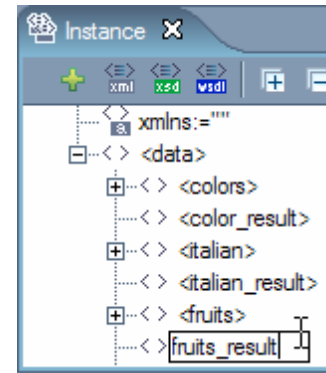
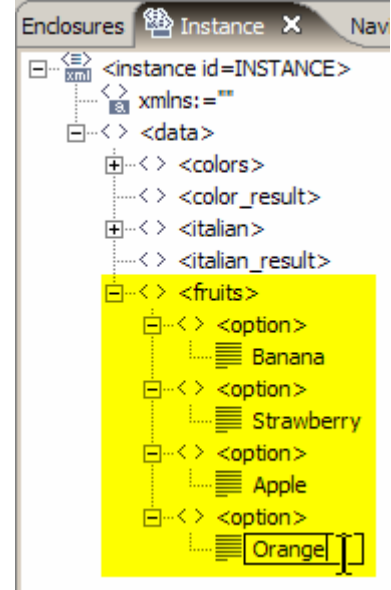




### Add a combobox to the form



1. Create a child called **fruits** to the instance created in step 1. Create an **option** for **Banana, Strawberry, Apple and Orange**.
2. To create a **combobox** from a data instance, use the select1 control.
  - a. Left click on the **Combobox (Select1)** item from the Standard Library on the Palette.
  - b. Left click on the canvas to place the list select1 control on the form.
3. Set the **label** to **'What is your favorite fruit?'**
4. Expand the XForms control in the properties of the popup item.
5. In the **Instance View** select one of the option elements under "fruit", right-click and select **Copy Reference**.
  - a. In the **Properties View**, expand XForms and paste the text into the value of the **nodeset** option.
  - b. Set the **itemset > value > ref** and **itemset > label > ref** to '...'.
6. Add another node called **fruits\_result** to the instance to store the result of the choice.
7. Now we will link the **fruits\_result** node we just created to the favorite fruit popup. Click on the popup in the canvas. In the properties view, expand the XForms (select1) properties and set



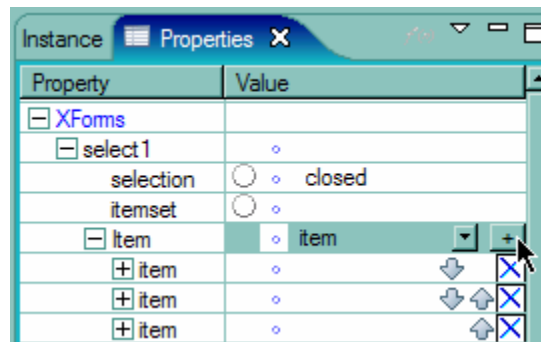
the **ref** property to **instance('INSTANCE')/fruits\_result**.

8. Preview the form.

Add a RadioGroup to the form

1. To create a radio group from a data instance, use the **select1** control.
  - a. Left click on the **RadioGroup (Select1)** item from the Standard Library on the Palette.
  - b. Left click on the canvas to place the radio group control on the form.
2. Set the value of the **label** property to **'What is your favorite bird of prey?'**
3. Expand the XForms control in the properties of the popup item.
4. This time instead of using itemset we are going to specify the items in the list explicitly. Expand the XForms category in the Properties View, set the itemset back to default (**click the blue dot so that it is clear**).

- a. Expand **Item** and add three item children by clicking on the '+' button three times.



- b. Expand each of the children and set the label of each item to **'Eagles'**, **'Hawks'**, and **'Falcons'** – these are the labels that will be shown to the user when the form is previewed.
- c. Now repeat the previous step, however this time set the value of each item instead of the label. The value is what is stored, based on the user's selection, in the xforms instance node that is linked to the radiogroup.

5. Add another node to the instance to store the result of the choice called "birds\_result".
6. Link the instance element "birds\_result" to the group of radio buttons by selecting the instance element in the **Instance View** and dragging it onto the RadioGroup object. This action will set the ref of the radiogroup "instance('INSTANCE')/birds\_result".

ref instance('INSTANCE1')/birds\_result

7. Preview the form.

### XFORMS Items

What is your Favorite Color? ▼

What is your favorite Italian

Pizza  
 Lasagna  
 Spaghetti  
 Calazone

What is your favorite fruit?

▼

What is your favorite bird of

Eagles

Hawks

Falcons

## Add an Attachment Button to the form

Note: Only a single file may be attached using XFORMS attachment procedures.

1. First we need to create an xml node that will hold information about the file. The data instance that will contain the attached file must include the following nodes:

**Data node** - The node that will contain the actual file you are attaching. Call this node filedata.

**filename** — The node that will contain the actual name of the file.

**mediatype** — The node that describes the file's mediatype.

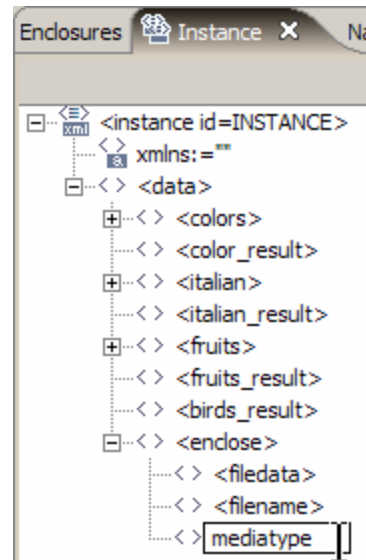
2. To create an attachment button from the data instance created above.



- a. Left-click on the **Button (Upload)** item from the Standard Library on the Palette.
- b. Left-click on the canvas to place the button on the form.

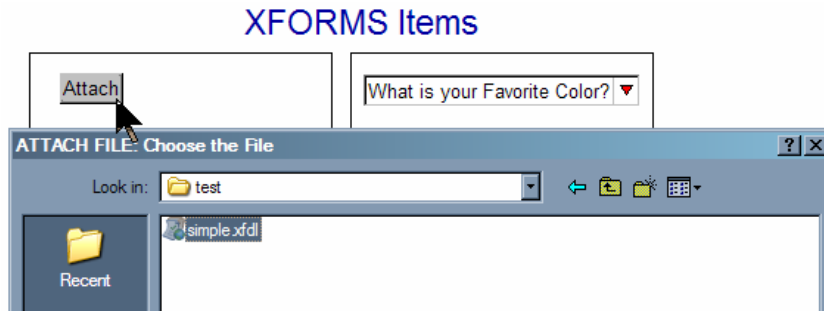
3. Set **XForms > label > Text** to **'Attach'** as shown below.

4. Set **XForms > ref** to the path to the data node which is **instance('INSTANCE')/enclose/filedata**. This attribute links the upload control to the node that will contain the file.



Property	Value
<b>XForms (upload)</b>	
mediatype	<input checked="" type="radio"/> */*
id	<input type="radio"/>
<b>label</b>	
Text	<input type="radio"/> Attach
ref	<input type="radio"/>
bind	<input type="radio"/>
model	<input type="radio"/>
alert	<input type="radio"/>
help	<input type="radio"/>
hint	<input type="radio"/>
Actions	<input type="radio"/> <empty>
ref	<input checked="" type="radio"/> instance('INSTANCE1')/enclose/filedata

5. Set the **XForms > mediatype > ref** and **XForms > filename > ref** attributes to the paths of their corresponding instance nodes (ie. **instance('INSTANCE')/enclose/mediatype** and **instance('INSTANCE')/enclose/filename**). These attributes link the mediatype (ie. text/plain) and filename (ie. sample.txt) properties of the selected file to the instance nodes we created.
6. Preview the form.



### Add an Input Field to the form

1. Create a node to store the input field data. Call it **field\_data**.

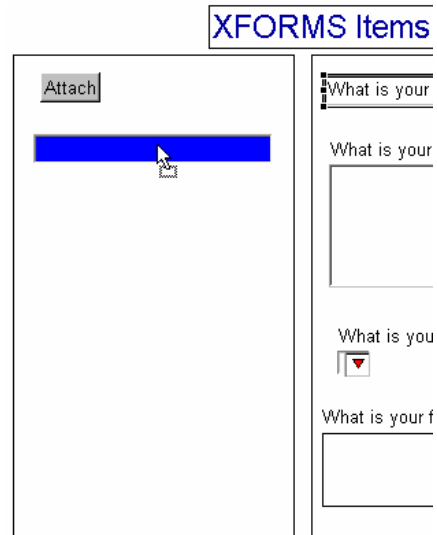
```
<> <fruits_result>
<> <birds_result>
+<> <endose>
<> field_data
```

2. To create a field:



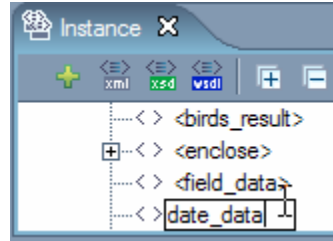
- a. Left-click on the **Field (Input)** item from the Standard Library on the Palette.
- b. Left-click on the canvas to place the list select1 control on the form

3. To link the data node with the field, left-click on the data node 'field\_data' and drag it over the input field on the canvas. Release the mouse.
4. The background color and text color may be changed by changing the **fontcolor** and **bgcolor** attributes in the properties dock.



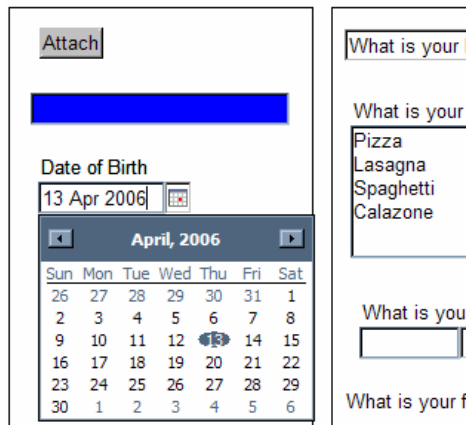
### Add Date Picker to the form

1. Create a child node to data for storing the result of the date picker. Call it **date\_data**.



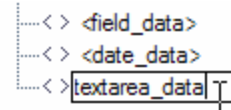
2. To create a field:
  - a. Left click on the **Date Picker (Input)** item from the Standard Library on the Palette.
  - b. Left click on the canvas to place the Calendar control on the form.
3. Change the value of the **label** property to **'Date of Birth'**.
4. To link the data node with the date picker, left click on the data node 'date\_data' and drag it over the date picker on the canvas. Release the mouse.
5. Preview the form.

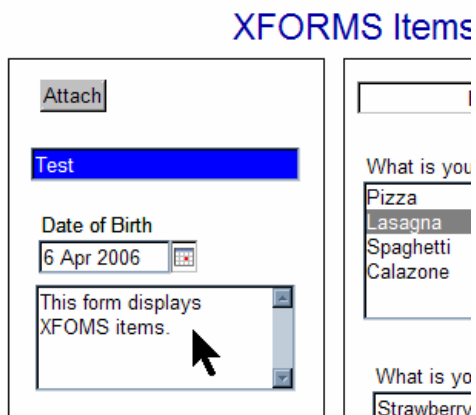
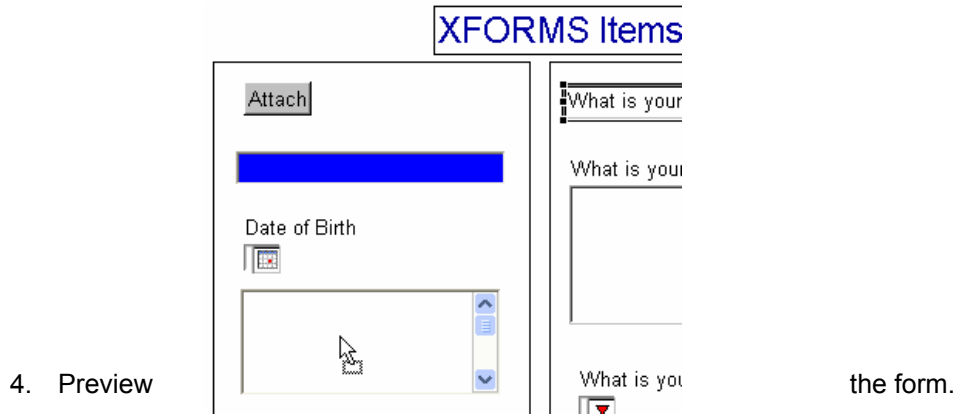
### XFORMS Items



#### Add a Text Area to the form

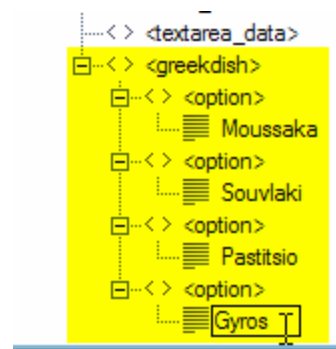
1. Create a node to store the input field data. Call it **textarea\_data**.
2. To create a text area:
  - a. Left click on the **Field (Text Area)** item from the Standard Library on the Palette.
  - b. Left click on the canvas to place the Text Area control on the form
3. To link the data node with the field, left click on the data node 'textarea\_data' and drag it over the input field on the canvas. Release the mouse.



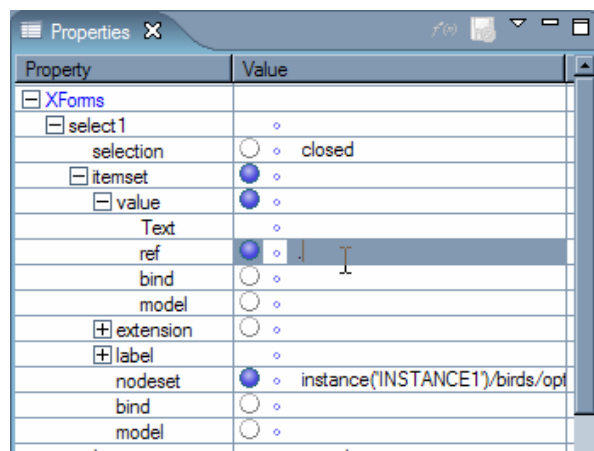


**Add a Mutually Exclusive Checkbox Group to the form**

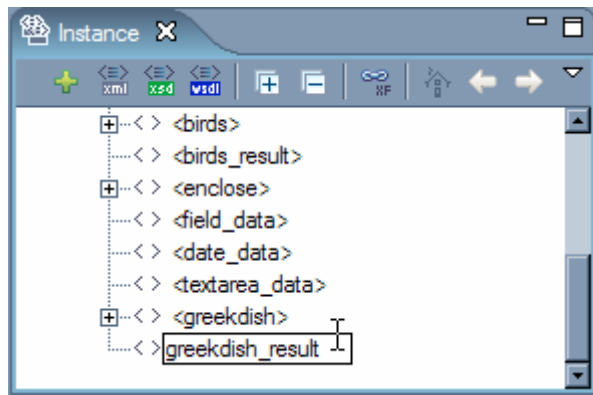
1. Create a node called **greekdish** as a child of data element. Create an option for **Moussaka, Souvlaki, Pastitsioto and Gyros.**
2. To create a check group from a data instance:
  - a. Use the select1 or select control. The select1 control limits users to a single choice.
  - b. Left-click on **CheckGroup (Select1)** item from the Standard Library on the Palette.
  - c. Left-click on the canvas to place the list select1 control on the form.
  - d. Set the **label** to **'What is your favorite Greek dish?'**.



3. Expand the XForms control in the properties of the popup item.
4. **Itemset** allows you to create lists and groups that offer dynamic choices that are determined while the form is in use. Itemset



- has one attribute called Nodeset, which is the path to the group of nodes that provide the list options. The label control is a mandatory element of the itemset control.
5. This itemset **label** is used to provide the choices for the list. Using a reference to the attributes of the <option> nodes, label creates and displays the list of choices that are displayed to the user.
  6. The itemset **value** control indicates which options are selected. Value has one attribute called ref, which is the path to the instance node.
  7. Set the **itemset > value > ref** and **itemset > label > ref** to '.' and the **nodeset** to **instance('INSTANCE')/greekdish/option**.
  8. Add another node to the instance to store the result of the choice.
    - a. Set **Select1 > ref** to **instance('INSTANCE')/greekdish\_result**.



9. Preview the form.

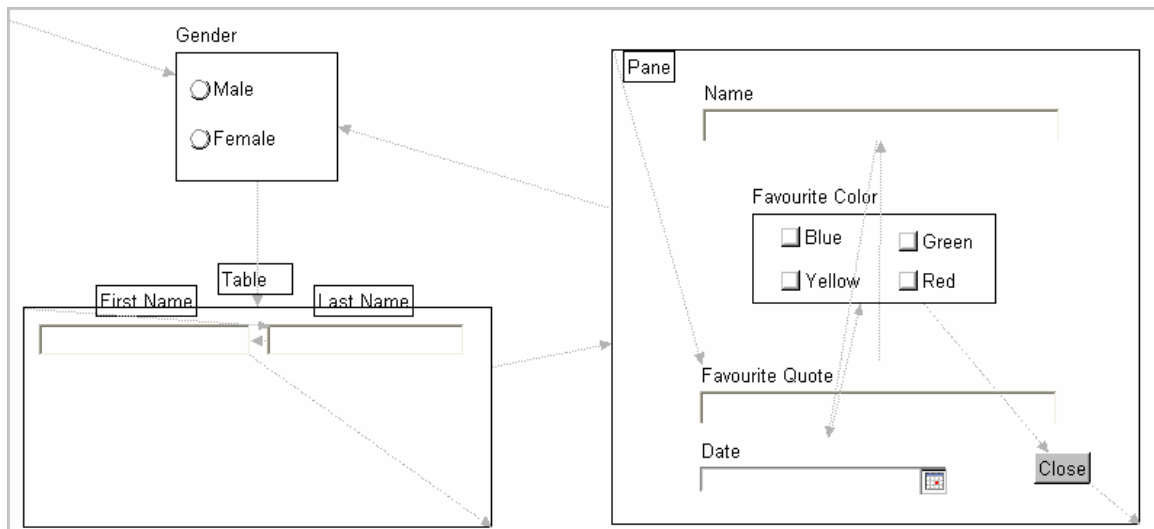
### XFORMS Items

<p>Attach</p> <p>[Redacted]</p> <p>Date of Birth 6 Apr 2006</p> <p>This form displays XFORMS items.</p> <p>What is your favorite greek dish?</p> <p><input type="checkbox"/> Moussaka</p> <p><input type="checkbox"/> Souvlaki</p> <p><input checked="" type="checkbox"/> Pastitsio</p> <p><input type="checkbox"/> Gyros</p>	<p>What is your Favorite Color? ▼</p> <p>What is your favorite Italian</p> <p>Pizza</p> <p>Lasagna</p> <p>Spaghetti</p> <p>Calazone</p> <p>What is your favorite fruit?</p> <p>Apple ▼</p> <p>What is your favorite bird of</p> <p><input type="radio"/> Eagles</p> <p><input checked="" type="radio"/> Hawks</p> <p><input type="radio"/> Falcons</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 5.9. Tab Order

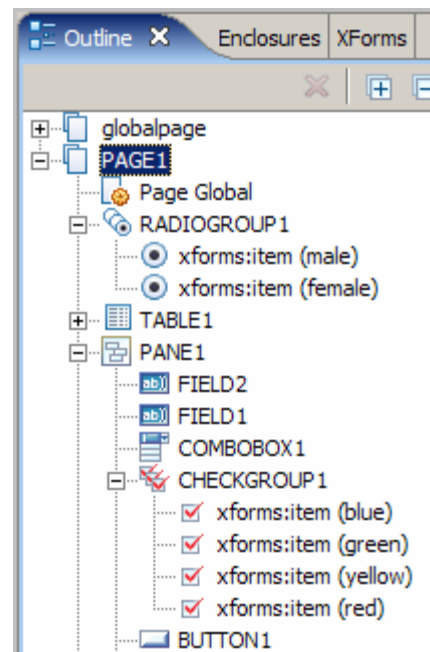
When creating an electronic form, an important final check is to determine if the tab order is correct. Workplace Forms Designer 2.6 has a number of methods for handling tab order: manually moving the tab arrows, using the 'Connection Creation Tool', explicitly defining the next and previous items in the tab order through an item's property view or modifying the build order. To enable the Connection Creation Tool, which appears in the palette, the tab order view must be visible.

The way to handle XForms item's tab order is similar to the way XFDL handles tab order. One difference to make note of is how XForms handles the tab order within groups. Notice there are no grey arrows between the Male and Female radio button in the RadioGroup item. This is how XForms handles grouped items. The tab order is handled through the build order. When we look at the Gender RadioGroup and the Favorite Color CheckGroup in the Outline tab below, the build order of the XForms items are displayed.



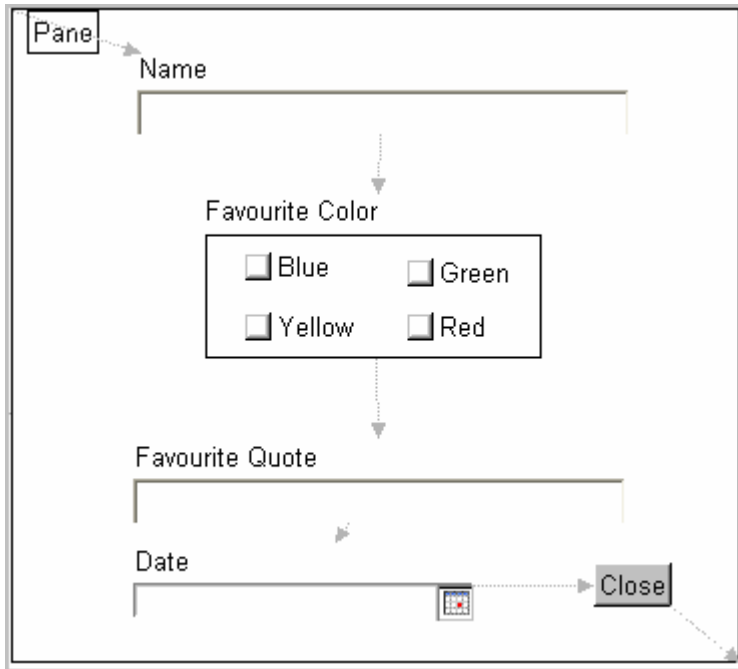
Using RADIOGROUP1 as an example, the XForms item male will gain focus when the RadioGroup first gains focus. When the user tabs to the next item, the XForms item, female, will gain the focus.

The Table and Pane objects are treated like an item, yet within these objects the tab order can be defined. The tab order within these XForms objects are managed the same way XFDL items are ordered, through the build order, Connection Creation Tool, or manually modifying the tab order.





These XForms objects have a unique linker, an initial pointer that points from the object to the item within the object and then a final pointer from the last item to the object. In the example below, the initial pointer points from the Pane object to the Name field. The final pointer points from the Close button to the Pane object.



Note: You can not point an item within an XForms object to an item outside of the object.

## 5.10. XForms Exercise 2 – Modifying the Tab Order

The tab order will go from left to right and from top to bottom for this exercise.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
  - b. Switch to the Resource Perspective and open **XForms\_Ex02\_XFormsTabOrder.xfd** from the **TrainingExercises\_XForms** project in the Navigator View.
2. To view the tab order, select **View > Show Next Tab Order**
3. Adjust the tab order to go from left to right, top to bottom using one of the methods listed below.

### **Moving the tab arrows**

- a. Select an arrow (back squares will appear at both ends of the arrow).
- b. Position mouse over one end until a cross appears.
- c. Click and drag the arrow to reflect the correct tab order .

### **Using the ‘Connection Creation Tool’**

- a. Select the Connection Creation tool from the palette, (next/previous tab order must be visible for the Connection Creation tool to be enabled).
- b. Select an item on the Designer canvas, this will be your starting point. Select a second item that is to follow the first in the tab order. An arrow will be drawn to show the connection.

### **Explicitly setting tab order through an items Property view**

- a. Select the Property view for an item.
- b. Under the General heading, set the next and previous values either by entering an items sid or by using the dropdown list.

### **Modifying the Build Order**

- a. Select the Outline view for an item.
- b. Drag and drop the item within the Outline view to reflect the proper tab order.

## 6. Understanding XPath

### 6.1. Audience

This section is intended to teach the user the fundamentals of the XPath language, which comprises most of the expressions written in XForms. This module will help you to understand and write your own XPath expressions; enabling you to build dynamic XForms forms.

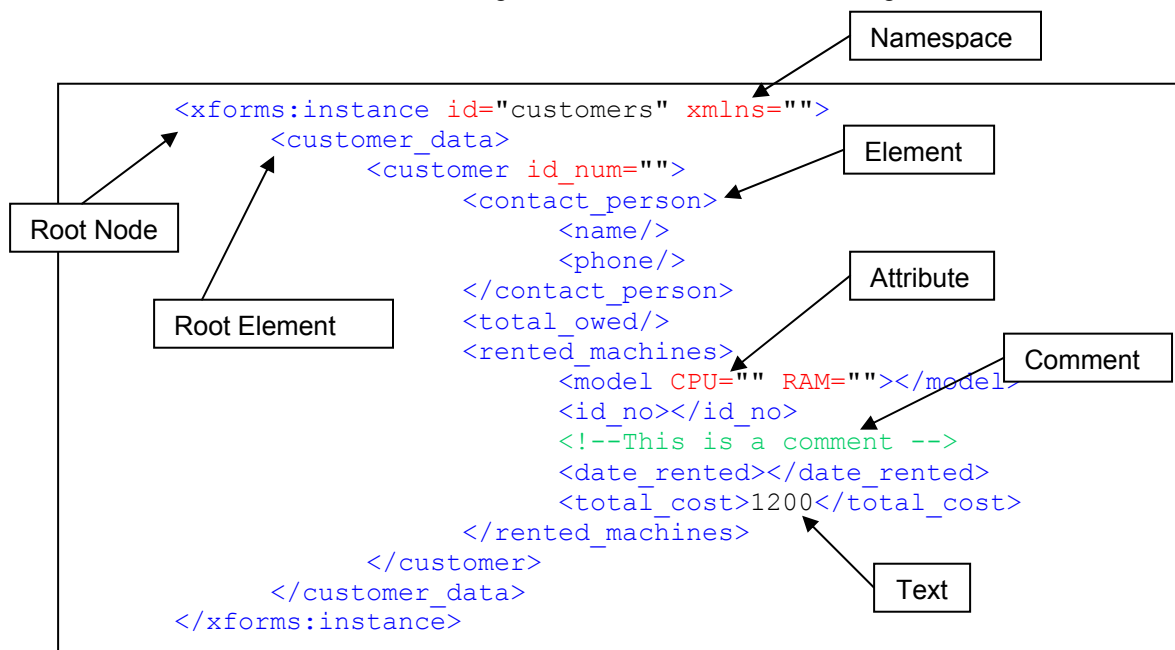
### 6.2. Overview

XPath is a language for referencing data elements (nodes) in an XML instance. XPath can be used to navigate through elements and attributes of an XML instance. Many of the statements in XForms are XPath expressions; therefore learning to read, write and interpret XPath expressions is very important.

- XPath is a syntax for referencing parts of an XML instance
- XPath uses path expressions to navigate XML instances
- XPath contains a library of standard functions
- XPath is a W3C Standard

**Note:** The term refers to an element of an XML instance. Each tag within an instance is a node. Nodes can have children, which are also referred to as nodes.

An XML instance is a tree comprised of a root node that may contain multiple levels of other nodes. From the XPath perspective there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes (there is a difference between the root node and the root element). The tree's root node contains the entire document, including the root element and comments and processing instructions that occur before the root element start tag or after the root element end tag.



## 6.3. Namespaces

Namespaces are an important piece in the XML document puzzle and XForms documents are no different. XForms introduces several new namespaces into the mix:

```
xmlns:ev="http://www.w3.org/2001/xml-events"  
xmlns:xforms="http://www.w3.org/2002/xforms"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Each of these namespaces is required within an XForms document. They are responsible for adding functionality to our XForms implementation.

**xmlns:ev** – Allows us to access XForms events like DOMActivate, xforms-value-changed, and xforms-submit-done.

**xmlns:xforms** – This is the XForms namespace where all the XForms specific tags are defined (i.e. xforms:input, xforms:output, etc.).

**xmlns:xsd** – This is where all the tags and options are defined for defining an XML Schema.

**xmlns:xsi** – This is where all the tags and options are defined for defining an XML Schema Instance.

## 6.4. XPath Operators

### 6.4.1. Numeric

Operand	Description
+	Addition
-	Subtraction
*	Multiplication
div	Division
mod	Modulus (returns the remainder)

### 6.4.2. Boolean

Operand	Description
or	Boolean Or
and	Boolean And
=	Boolean Equal
!=	Not Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

## 6.5. XPath References

### 6.5.1. Building XPath References

The majority of XPath expressions are used to select a node or set of nodes from an XML instance. When an XPath expression returns a node or set of nodes the expression is referred to as a location path. The XPath location path is the most common type of XPath expression. Location Paths can be relative or absolute; a relative path includes the reference from the current node and an absolute path includes the complete path (from the beginning of the document). The following is a table of the basic syntax elements used for creating XPath expressions that return location paths.

Expression	Description
<code>instance('instanceID')</code>	Selects the instance whose id matches “ <i>instanceID</i> ”, if an “ <i>instanceID</i> ” is not specified then the first instance is used as the “default”
<code><i>nodename</i></code>	Selects all nodes, in the default instance, whose name matches “ <i>nodename</i> ”
<code><i>/nodename</i></code>	Selects all nodes, in the default instance, whose name matches “ <i>nodename</i> ” and they are root nodes (nodes at the first level of the instance)
<code><i>//nodename</i></code>	Selects all nodes, anywhere in the default instance, whose name matches “ <i>nodename</i> ”
<code>.</code>	Selects the current node
<code>..</code>	Selects the parent of the current node
<code>@<i>attributeName</i></code>	Selects all attributes, where name matches “ <i>attributeName</i> ”
<code><i>nodename</i>[<i>exp</i>]</code>	Selects all the nodes whose name matches “ <i>nodename</i> ” and where the “ <i>exp</i> ” evaluates to true. Referred to as a predicate.

### 6.5.2. The Root Location Path

The simplest location path is one that selects the root node of the XML instance, which is defined by forward slash (`/`). The forward slash is an absolute location path because no matter where you are when you use it, it always refers to the root node.

### 6.5.3. Locating Children Elements

Selecting children elements can be done by first specifying the root node and then the desired child (`/bookstore`). This command will return all the nodes that are children of the root node and called “bookstore”.

### 6.5.4. Sample XForms Instance

```

<xforms:instance id="INSTANCE" xmlns="">
  <data>
    <bookstore>
      <book lang="en">
        <title>Book1</title>
        <author>Jane Doe</author>
        <price>9.99</price>
      </book>
      <book lang="en">
        <title>Book2</title>
        <author>Scott Tiger</author>
        <price>35.99</price>
      </book>
      <book lang="en">
        <title>Book3</title>
        <author>Joe Nobody</author>
        <price>69.00</price>
      </book>
      <book lang="fr">
        <title>L'auvre4</title>
        <author>Pierre Francois</author>
        <price>38.00</price>
      </book>
    </bookstore>
  </data>
</xforms:instance>

```

### 6.5.5. Locating Attributes

Attributes can also be referenced by using XPath. Use the “at” sign (@) followed by the name of the attribute to retrieve its contents.

```
/bookstore/book[1]/@lang
```

This expression will return the “lang” attribute for the first book in the bookstore. In XPath arrays begin with an index of 1.

### 6.5.6. Predicates

Predicates are used to find a specific node or set of nodes that meet a specified criterion. Predicates are defined by using square brackets. The expression used in the value of a predicate must evaluate to either ‘true’ or ‘false’. If you want to dynamically determine the position you must use the position() function in the expression.

Common predicate examples are described in the following table:

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child

	of the bookstore element
<code>/bookstore/book[last() - 1]</code>	Selects the second to last book element that is the child of the bookstore element
<code>/bookstore/book[position() &lt; 3]</code>	Selects the first two book elements that are children of the bookstore element
<code>//book[@lang]</code>	Selects all the book elements that have an attribute named lang
<code>//book[@lang='en']</code>	Selects all the book elements that have an attribute named lang with a value of 'eng'
<code>/bookstore/book[price &gt; 35.00]</code>	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
<code>/bookstore/book[price &gt; 35.00]/title</code>	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

### 6.5.7. XPath Sample Form

A form has also been provided that demonstrates these XPath expressions in action. Open `Sam_XFormsPredicates.xfdl` from the training package. Several of the expressions described above have been demonstrated in this form. Each section contains an `xforms:repeat` that is linked to the sample XForms instance using XPath predicate statements. Feel free to experiment with predicates within this sample form.



## 6.6. XPath Functions

The following is a table of the XPath functions that are supported.

Function	Parameters	Description
Count	Reference to Instance Element	Returns the number of children elements
String-length	String or Reference to Instance Element	Returns the length of the string
Local-name	Reference to Instance Element	Returns the local name of the reference
Name	n/a	Returns the element name of the corresponding nodeset
Number	Reference to Instance Element	Converts the item to a Number
Ceiling	Reference to Instance Element	Rounds the number up to the nearest value
Floor	Reference to Instance Element	Rounds the number down to the nearest value
Round	Reference to Instance Element	Rounds the number to the nearest value
String	Value or Reference to Instance Element	Converts the item to a string
Concat	Comma separated values to concatenate together	Combines multiple strings together to form one string
Substring	Source String, Start value, End value	Returns the characters from the start to the end values of the source string.
Contains	Source String, Search String	Returns true if the source string contains the search string
Starts-with	Source String, Search String	Returns true if the source string begins with the search string
Normalize-space	Source String or Reference to Instance Element	Removes all excess White space
Substring-before	Source String, Search String	Returns the characters of the source string that precede the first

		occurrence of the search string
Substring-after	Source String, Search String	Returns the characters of the source string that follow the first occurrence of the search string
Boolean	XPath Expression	Converts item to a Boolean
Not	XPath expression	Provides the negative of the expression
True	n/a	Returns a Boolean "true"
False	n/a	Returns a Boolean "false"

### 6.6.1. XPath Functions Sample Form

A form has been provided that demonstrates each of the above functions being used. The form is titled "Sam\_XPath Functions.xfdl" and can be found in the package distributed with this document.

## 7. Advanced XForms Concepts

### 7.1. Audience

This section is intended for users who must understand how to build dynamic forms with an XForms model.

### 7.2. Overview

XForms can be a difficult language to learn, it may be an emerging standard but it is still very much in its infancy. This section is focused on presenting some of the more complex concepts of XForms. After reading this section the reader should:

- Understand how to create a bind in the Designer to manipulate instance data
- Understand how to use XForms actions
- Understand how to use XForms events

### 7.3. Adding Form Logic

There are two types of form logic; rules that define data constraints and rules that define how the form will react to data input. Both XForms and XFDL have their own mechanisms for adding logic to control how the form behaves. Choosing which method to use will depend on what the rules are, how they affect the form, and whether or not the language supports it.

A general rule to follow, when building an XForms form, is that any logic dependent on the data should be implemented using XForms. Because XForms is a new technology there may be things that you want to do that are not currently supported, this is where XFDL steps in. Use XFDL to complement your XForms logic.

### 7.4. XForms Bind

In XFDL calculations are executed at the item level by adding a compute attribute to one of its options, but in XForms they are executed at the instance level. All of the XForms logic is triggered by changes in an XML instance, we use a bind to monitor an instance element and execute a calculation when the value changes.

An XForms bind must define a nodeset. A nodeset is the path to the instance element that is being manipulated. Once the nodeset has been defined then all subsequent attribute references become relative to the nodeset value.

There are several different properties of a bind that can be used to affect the value, behavior or appearance of an instance element and its associated form item. A bind may contain one or more of these attributes at the same time.

#### 7.4.1. readonly

The value of this attribute supports an XPath expression that should evaluate to true or false. Setting an element to readonly will not allow the value to be changed, either by the user or programmatically.

```
<bind nodeset="instance('personData')/people/person/name"
      readonly="true()"/>
```

### 7.4.2. required

The value of this attribute supports an XPath expression that should evaluate to true or false. Setting an element to required means that the user must enter a value into the corresponding form item before the form will be considered complete. The behavior of the form widget also changes, when the form loads the item will be a golden yellow, once the item is given a value then it will change to white.

```
<bind nodeset="instance('personData')/people/person/name"
      required="true()"/>
```

### 7.4.3. relevant

The value of this attribute supports an XPath expression that should evaluate to true or false. If an element is set to relevant then it will be included in a submission and the corresponding form element will be visible. If an element is not relevant, then it will not be included in a submission, and the corresponding form item will be invisible.

```
<bind nodeset="instance('personData')/people/person/name"
      relevant="true()"/>
```

### 7.4.4. calculate

The value of this attribute supports an XPath expression. The calculate attribute supports standard mathematical operations as well as references to instance data. This is most commonly used for calculating the result of evaluating multiple instance elements. An example of a calculate attribute is:

```
<bind calculate="instance('itemData')/item/cost *
               instance('itemData')/gst" />
```

### 7.4.5. constraint

The value of this attribute supports an XPath expression that should evaluate to true or false. If the constraints are not met, then the items in the nodeset will be flagged as containing invalid input.

```
<bind nodeset="instance('personData')/people/person/age"
      constraint=". > '18'"/>
```

For a detailed description of the properties described above, please refer to the “Details on XForms Options -- xformsmodels” section of the XFDL Specification.

### 7.4.6. Creating a Bind

To create a bind in the Designer:

1. Open the XForms View, expand “XForms” and “Model: Default”.

2. Then right-click “Model: Default” and select **Create Bind** from the menu.
3. Once the bind appears in the **XForms View**, select it and open the **Properties View**.
4. Expand “General” and “Model Item Properties”.
5. Within the “Model Item Properties” category you will find all the attributes described above; select required and enter “true()” as its value.
6. The last step is to define the nodeset, set it to `instance('personData')/person/name`. Now if the field has been “linked” to the instance data, then when you preview the form it should appear with a yellow background.

## 7.5. XForms Exercise 3 – Implementing XForms Binds

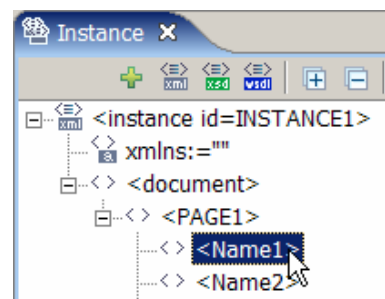
This lesson is meant to introduce you to the way XForms binds are created in the designer. In XForms, binds are used to handle logic in the form based on the data stored in the XForms model's nodes. Internally, XPath is used to define the functions and nature of the calculation to be performed. Since the goal of this exercise is to gain familiarity with the designer and not to learn XPath, we have provided the necessary lines of code for you. This exercise will expose you several forms of binds.

This document has been written assuming little to no previous knowledge of XFDL, XForms and XPath. This tutorial assumes that the reader has some basic understanding of the Eclipse interface.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Switch to the Resource Perspective and open **XForms\_Ex03\_XFormsBinds.xfd** from the **TrainingExercises\_XForms** project in the **Navigator View**.
3. First Calculation: Setting a value equal to the value of another item.

Equal to the value of an item	
Name	Copy of name
<input type="text"/>	<input type="text"/>

- a. In order to “copy” data from one field to another we will simply link the two fields to the same node in the instance.
  - i. To do this you will have to drag and drop the `<Name1>` element in the instance view onto **FIELD2** (below **Copy of name**).
  - ii. Since both fields are linked to the same instance element this will simulate copying data from one element to another



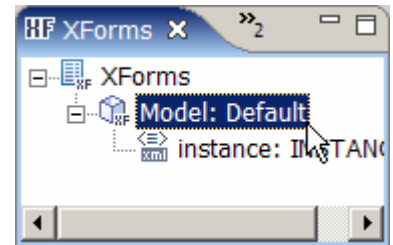
Note: the readonly property of the **Copy of name** field must be set “on”, otherwise both fields can edit the text that is stored in the node.

+ XForms (input)	
- General	
sid	o NameCopy
label	o
active	o <input checked="" type="checkbox"/> on
previous	o
next	o
readonly	<input checked="" type="checkbox"/> on

- b. If you want to test your work click on the **Preview** tab to load the form.
    - i. Try inputting a value into the **Name** field (FIELD1) and see if it gets copied over to the **Copy of name** field (FIELD2).
    - ii. It's important to note that the changes won't occur until the focus leaves FIELD1 (i.e. tabbing or mouse clicking onto another field would change the focus).
  - c. **BONUS question:** Can you get the same result by creating a bind and linking FIELD2 to the bind instead of using a ref? If you don't know the answer try the rest of the exercise and come back to it.
4. Calculation Bind: Set a field by a calculation of two values.

Set by a calculation of two values		
Multiplicand	Multiplier	Product
<input type="text"/>	<input type="text"/>	<input type="text"/>

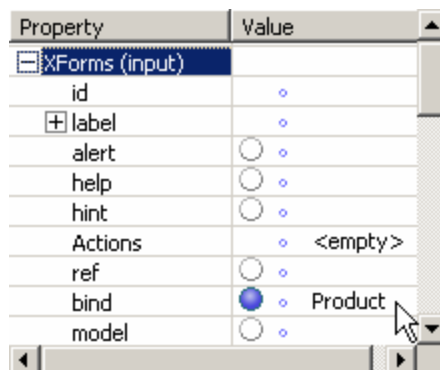
- a. Open up the **XForms View** and right-click **Model: Default** and select **Create Bind**. Then open up the **Properties View** of the bind by left clicking on the newly created bind.
- b. We've chosen to do a multiplication of two integers as our calculation.



- i. First we have to give our new bind an id (Product was chosen for this example) then we need to formulate the correct XPath statements.
- ii. The **nodeset** needs to point to the Product element while **calculate** has to contain the two elements being multiplied together.
- iii. If you want to try this out on your own then don't look at the screen shot below; otherwise, fill in the **id**, **calculate**, and **nodeset** fields as shown below.

[-] Model Item Properties	
type	<input type="radio"/> ◦
readonly	<input type="radio"/> ◦
required	<input type="radio"/> ◦
relevant	<input type="radio"/> ◦
calculate	<input checked="" type="radio"/> ◦ ../Multiplicand * ../Multiplier
constraint	<input type="radio"/> ◦
[-] General	
nodeset	<input checked="" type="radio"/> ◦ instance('INSTANCE1')/PAGE1/Product
id	<input type="radio"/> ◦ Product

- c. Now link the bind to the appropriate input field by clicking on FIELD13 (below **Product**) and selecting the **Product** bind as shown below.



- d. If you want to test this calculation select the **Preview** tab and enter an integer into both the **Multiplicand** and **Multiplier** fields and then press tab. The **Product** field should contain the product of the two values.

5. Calculation Bind: The sum XPath function.

There are many XPath functions that can be used in the bind properties. Here we will use the `sum(xpath1, xpath2,...)` function which takes a number of xpath arguments pointing to nodes. It then sums up the values contained in all those nodes. Note that the xpath arguments can each point to *nodesets*, in which case all the nodes in the nodesets will be summed. We will use this property in the exercise to provide a single xpath to all the nodes we want to sum.

Equal to the sum of multiple fields			
A	B	C	Total
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

- Open up the **XForms View** and right-click **Model: Default** and select **Create Bind**. Then open up the **Properties View** of the bind by left clicking on the newly created bind.
- Open up the **Properties View** for the bind by left clicking on the newly created bind.
- In order to find the sum of these three elements we will have to create the proper XPath statement.
  - For this sample bind we've chosen "Sum" as the id.
  - If you want to attempt the XPath statements on your own then ignore the screenshot below; otherwise, fill in the **id**, **calculate**, and **nodeset** fields as shown.



Property	Value
[-] Model Item Prope...	
type	<input type="radio"/> ◦
readonly	<input type="radio"/> ◦
required	<input type="radio"/> ◦
relevant	<input type="radio"/> ◦
calculate	<input checked="" type="radio"/> ◦ sum(../Numbers/Int)
constraint	<input type="radio"/> ◦
[-] General	
nodeset	<input checked="" type="radio"/> ◦ instance('INSTANCE1')/PAGE1/Total
id	<input type="radio"/> ◦ Sum

**XPath Discussion:** Note that “sum(../Numbers/\*)” and “sum(../Numbers/Int[1], ../Numbers/Int[2], ../Numbers/Int[3])” are equivalent to the solution shown above.

- d. Now link the bind to the appropriate input field by clicking on the Total field (below **Total** label) and selecting the **Sum** bind as shown below.

Property	Value
[-] XForms	
[-] input	◦
id	◦
[+] label	◦
alert	<input type="radio"/> ◦
help	<input type="radio"/> ◦
hint	<input type="radio"/> ◦
Actions	◦ <empty>
ref	<input type="radio"/> ◦
bind	<input checked="" type="radio"/> ◦ Sum
model	<input type="radio"/> ◦

- e. If you want to test the sum calculation:
- i. Click on the preview tab.
  - ii. Fill in fields **A**, **B**, and **C** with integer values then press tab.
  - iii. The sum of these 3 values should be located in the **Sum** field.
6. Setting Multiple Properties: The Required and Relevant Properties.  
So far, we have only dealt with a single bind property, the calculate property. In this exercise, we will demonstrate the use of the relevant and required properties. Also, we

will set both properties on the same bind. The goal is to make a field that will only be visible and required when a checkbox is checked.

### Multiple properties (required & relevant) set by checkbox

Check here if you have a phone number to enter Phone Number

- a. Open up the **XForms View** and right-click **Model: Default** and select **Create Bind**. Then open up the **Properties View** of the bind by left clicking on the newly created bind.
- b. Open up the **Properties View** for the bind by left clicking on the newly created bind.
- c. Now we need to set the bind properties so that the **Phone Number** field will only appear and be mandatory when the check box is checked.
  - i. For this example we've chosen "RequiredAndRelevant" as the bind **id**.
  - ii. Provide an xpath query for the **nodeset** that selects the "PhoneNumber" node in the data instance. See the screen shot below for the solution.
  - iii. Provide xpath queries for the **required** and **relevant** properties. Both these xpath queries will be the same. See the screen shot below for the solution.

Property	Value
<b>Model Item Properties</b>	
type	<input type="radio"/>
readonly	<input type="radio"/>
required	<input checked="" type="radio"/> ..//PhoneCheckbox = "true"
relevant	<input checked="" type="radio"/> ..//PhoneCheckbox = "true"
calculate	<input type="radio"/>
constraint	<input type="radio"/>
<b>General</b>	
nodeset	<input checked="" type="radio"/> instance(''INSTANCE1'')/PAGE1/PhoneNumber
id	<input type="radio"/> RequiredAndRelevant

**XPath Discussion:** We want to set the relevant and required properties to true whenever the checkbox is checked. So we need to write XPath queries which test the checkbox node. If an xpath statement returns any nodes, it is equivalent to the boolean value true. If it does not return any nodes, it is equivalent to boolean false. In this example the filterstep is [PhoneCheckbox = "true"] because we only want to select a node when the checkbox node is true.

- d. Now link the bind to the appropriate input field by clicking on the **TelephoneNumber** field (below **Phone Number** label) and selecting the **RequiredAndRelevant** bind as shown below.

Property	Value
<input type="checkbox"/> XForms (input)	
id	o
<input type="checkbox"/> label	o
alert	<input type="radio"/> o
help	<input type="radio"/> o
hint	<input type="radio"/> o
Actions	o <empty>
ref	<input type="radio"/> o
bind	<input checked="" type="radio"/> o RequiredAndRelevant
model	<input type="radio"/> o

- e. If you want to test the bind:
- i. Click on the preview tab.
  - ii. When the phone number check box is not checked, the field should be invisible. Click the checkbox.
  - iii. When the checkbox is checked, the field should appear and be colored yellow to indicate it is mandatory.
  - iv. Unchecking the checkbox should make the field disappear again.

## 7. Setting Constraints

In this exercise we will create a bind that will enforce a constraint on the data we enter into the form. In this case, that the number we enter is larger than the number in another field.

### Setting Constraints

Enter two numbers such that A is greater than B

A                      B

>

- a. Open up the **XForms View** and right-click **Model: Default** and select **Create Bind**. Then open up the **Properties View** of the bind by left clicking on the newly created bind.
- b. Open up the **Properties View** for the bind by left clicking on the newly created bind.
- c. Now we need to set the bind properties so that the **A** number field will show an error if the number entered in **A** is smaller than the number entered in **B**.

- i. For this example we've chosen "AGreaterThanB" as the bind **id**.
- ii. Provide an xpath query for the **nodeset** that selects the "NumA" node in the data instance. See the screen shot below for the solution.
- iii. Provide an xpath query for the **constraint** property that is true when the NumA node is greater than NumB. See the screen shot below for the solution.

Property	Value
<input type="checkbox"/> Model Item Properties	
type	<input type="radio"/> ◦
readonly	<input type="radio"/> ◦
required	<input type="radio"/> ◦
relevant	<input type="radio"/> ◦
calculate	<input type="radio"/> ◦
constraint	<input checked="" type="radio"/> ◦ . > ../NumB
<input type="checkbox"/> General	
nodeset	<input checked="" type="radio"/> ◦ instance('INSTANCE1')/PAGE1/NumA
id	<input type="radio"/> ◦ AGreaterThanB

**XPath Discussion:** The constraint xpath is also equivalent to "instance('INSTANCE1')/PAGE1/NumA > instance('INSTANCE1')/PAGE1/NumB", but here we used the xpath shorthand "." which means "current node" and ".." which means parent. XPath statements entered in the properties fields of a bind are relative to the xpath used to point to the nodeset, but you can also use full paths if you prefer.

- d. **BONUS** question: Can you predict what will happen if the nodeset for this bind points to NumB instead? Be careful to update the constraint xpath since it is relative to the nodeset (you can use the full paths provided in the XPath Discussion section).

## 7.6. XForms Actions

XForms actions allow you to initiate a number of processes, including submitting a form, setting a value in a form, and inserting a row in a repeat table.

XForms actions are triggered by events in the form. For example, you might create an action that occurs when the user clicks a button, when a particular value in the form has changed, or when a submission has returned an error.

Please refer to the “Details on XForms Actions” section of the XFDL specification for a complete list of all the actions that are supported and how to implement them.

## 7.7. XForms Exercise 4 – Using XForms Actions

In this exercise you will be introduced to the following four actions: `xforms:message`, `xforms:setvalue`, `xforms:setfocus`, and `xforms:setindex`.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6**
2. Switch to the Resource Perspective and open **XForms\_Ex04\_XFormsActions.xfd** from the **TrainingExercises\_XForms** project in the Navigator View.

### **xforms:message**

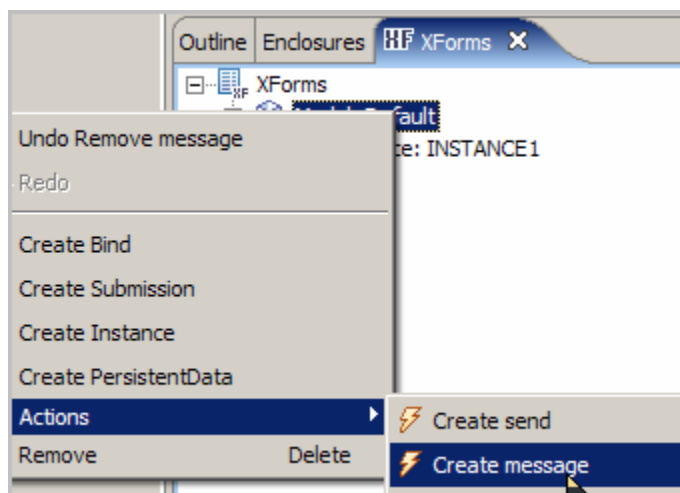
In this exercise we will show 2 different ways to use `xforms:message`. We will show a message box when the data instance has been initialized and when an XForms item's value has changed.

### **Data Model Method**

When creating the `xforms:message` in the data model, the code will look similar to this:

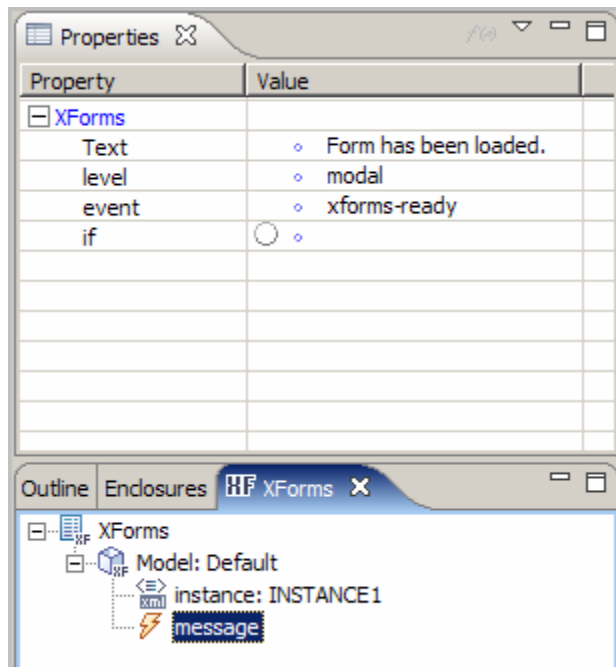
```
<xforms:message level="modal" ev:event="xforms-ready">Form has  
been loaded.</xforms:message>
```

1. Select the **XForms View**.
2. Right-click **Model:Default**.
3. Select **Actions > Create message**.



4. Highlight the newly created **message** item. Expand **XForms** in the Properties View.

5. In the **Text** property, input **Form has been loaded.**
6. In the **level** property, select **modal**. This setting is an XForms option which determines the style of the message box to be displayed. All messages are displayed as modal style message boxes by the viewer, so it doesn't matter which setting is chosen here.
7. In the **event** property, select **xforms-ready**. xforms-ready occurs when the forms viewing application has finished the initial set up of all XForms constructs and is ready for user interaction.
8. Your properties should look like the screen shot below.



9. Select the Preview tab and view results.

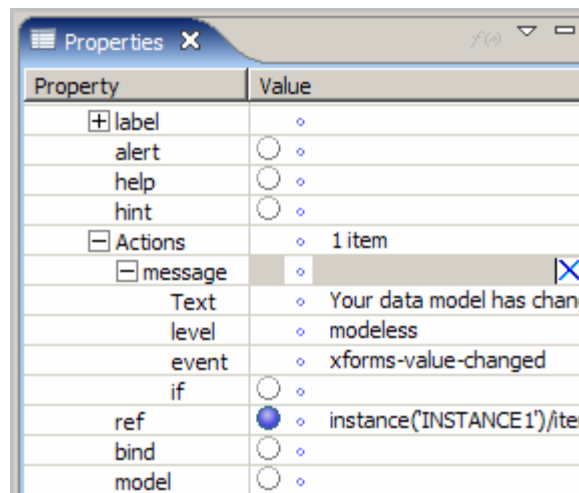
### Item Method

When creating an xforms:message in an item, the code will look similar to this:

```
<xforms:message level="modeless" ev:event="xforms-value-changed">Your data model has changed.</xforms:message>
```

1. Select the item, **FIELD1**.
2. Expand **XForms** in the Properties View.
3. Click **Actions**, select **message** in the drop down.
4. Once **message** is selected, click **+** to add the action.
5. Expand the **Actions** and **message** property.

6. In the **Text** property, input **Your data model has changed**.
7. In the **level** property, select **modal**. This setting is an XForms option which determines the style of the message box to be displayed. All messages are displayed as modal style message boxes by the viewer, so it doesn't matter which setting is chosen here.
8. In the **event** property, select **xforms-value-changed**. `xforms-value-changed` occurs when a value is changed in an `xforms:select` or `xforms:select1` option.
9. Your properties should look like the screen shot below.



10. Select the Preview tab to test the results.
11. Enter text into FIELD1 and tab out to see the message.

### **xforms:setvalue**

In this exercise, we will take the value found in FIELD1's data instance and set it to FIELD4's value using `xforms:setvalue`.

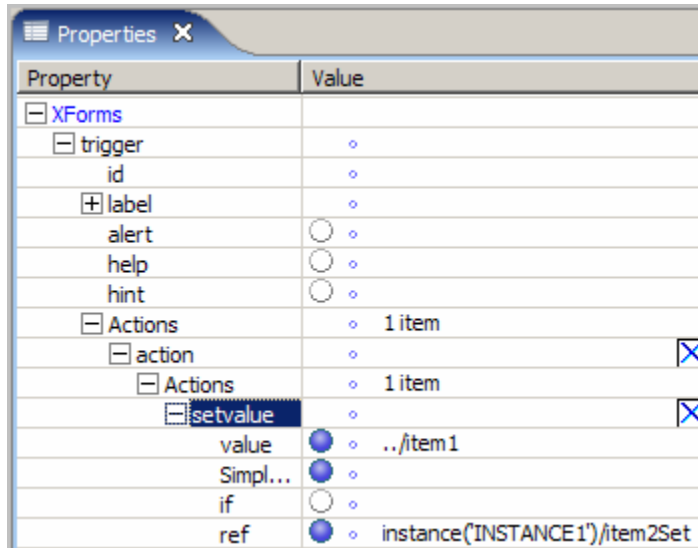
When completed, the source code in the `pushValue` button will look similar to this:

```
<xforms:setvalue ref="instance('INSTANCE1')/item2Set"
value="../item1"></xforms:setvalue>
```

1. Select the button, **pushValue**.
2. Expand **XForms (trigger) > Actions > action** in the Properties View.
3. Click **Actions**, select **setvalue** in the drop down.
4. Once **setvalue** is selected, click **+** to add the action.



5. Expand the **Actions** and **setvalue** property.
6. In the **value** property input `../item1`, hit **enter**.
7. In the **ref** property enter `instance('INSTANCE1')/item2Set`, hit **enter**. Alternatively you can locate item2Set in the instance view, right click on the node and select **Copy Reference**, then paste the reference into the ref property.
8. Your properties should look like the screen shot below.



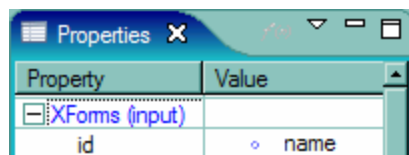
9. Select the Preview tab and view results.

### xforms:setfocus

In this exercise, we will set the cursor's focus back the Name field when the 'Reset Info' button is clicked. The values in the 3 fields will be cleared using the xforms:setvalue action which has already been setup.

The field that gains the focus will need to be given a unique id that the action's control will point to.

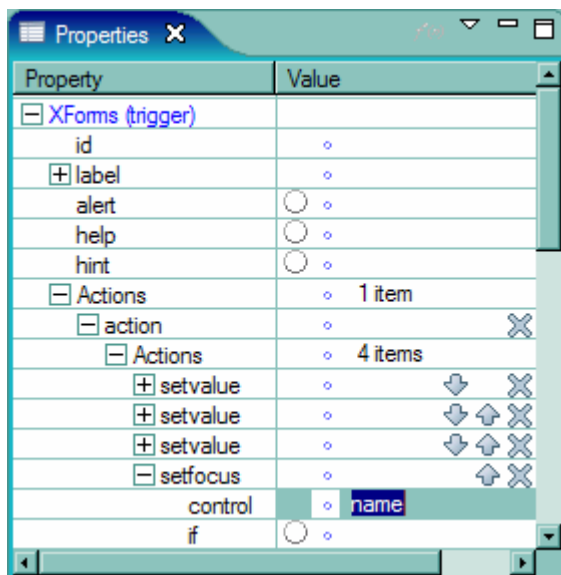
1. Select the field labeled "**Name**".
2. Expand **XForms (input)** in the Properties View.
3. In the **id** property, input the value **name**.



- The code view for `xforms:setfocus` will look like this when we are done.

```
<xforms:setfocus control="name"></xforms:setfocus>
```

- Select the button labeled "Reset Info" (the button's sid is "**resetFocus**").
- Expand **XForms (trigger) > Actions > action** in the Properties tab.
- Click **Actions**, select **setfocus** in the drop down.
- Once **setfocus** is selected, click **+** to add the action.
- Expand **Actions** and the newly created **setfocus** property.
- In the **control** property, input **name**, hit enter.
- Your properties should look like the screen shot below.




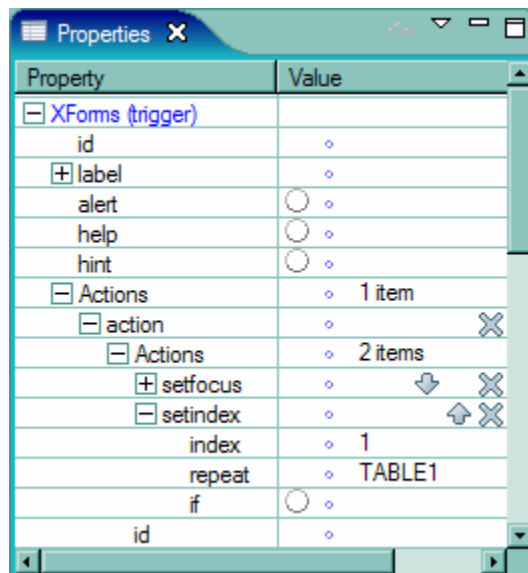
- Select the Preview tab and view results.

### **xforms:setindex**

In this exercise, `xforms:setindex` is used to set the focus to a particular index in a table. The table consists of three rows. When the button is selected, we set the index to the first row and then set the focus back to the table. The `setfocus` will already be provided; this exercise will concentrate on setting the index. The code will look similar to this:

```
<xforms:setindex index="1" repeat="TABLE1"></xforms:setindex>
```

1. Select the button labeled “Reset To This Index”, (the button’s sid is “**setindex**”).
2. Expand **XForms (trigger)** > **Actions** > **action** in the Properties View.
3. Click **Actions**, select **setindex** in the drop down.
4. Once **setindex** is selected, click  to add the action.
5. Expand **Actions** and the newly created **setindex** property.
6. In the **index** property, set the value to 1, hit enter.
7. Set the **repeat** property to the id of the XForms repeat object, which is “TABLE1”
8. Your properties should look like the screen shot below.



9. Select the Preview tab and view results.

## 7.8. XForms Events

XForms event handlers track events in the form, such as a button click or the selection of a particular choice. When these events occur, they are registered by the XForms system. This allows you to create actions that are triggered by these events. For example, you might create an action that is triggered when a particular button is clicked, or when a particular choice in a list is selected. Please refer to the “Details on XForms Event Handlers” section of the XFDL specification for a complete list of all the events that are supported and how to implement them.

## 7.9. XForms Exercise 5 – Using XForms Events

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6**
2. Switch to the Resource Perspective and open **XForms\_Ex05\_XFormsEvents.xfd** from the **TrainingExercises\_XForms** project in the Navigator View.

---

Note: If you are not using the example form to complete this exercise then after creating a blank form you must enable the **XForms** view. This can be done by clicking **Window > Show View > XForms**. Open the **XForms** tab in the **Designer** view, right-click anywhere inside the view and select **Add XForms Support**.

---

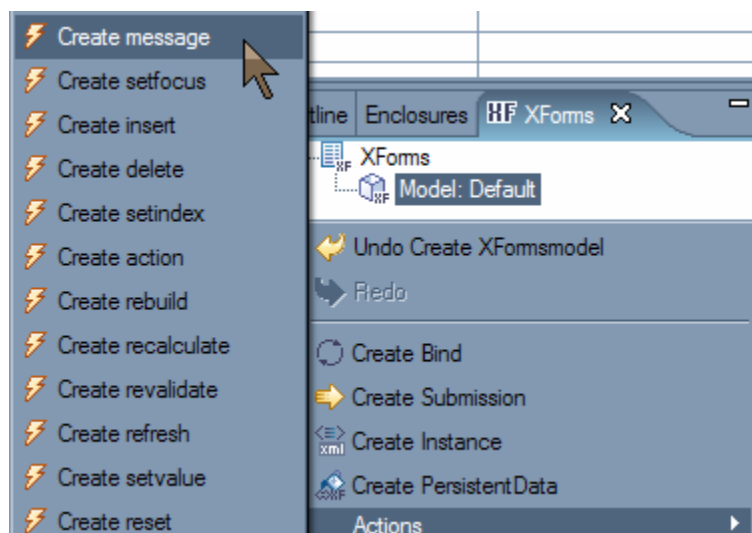
### Event 1 – XForms-Ready

This event occurs when the forms viewing application has finished the initial set up of all XForms constructs and is ready for user interaction. In this example this occurs before the form is displayed on the screen. In this part of the exercise, we will cause a message box to appear when the form has been initialized.

The code will look similar to this:

```
ev:event="xforms-ready"
```

1. Expand the XForms element and right click on **Model**. Select **Actions > Create Message**

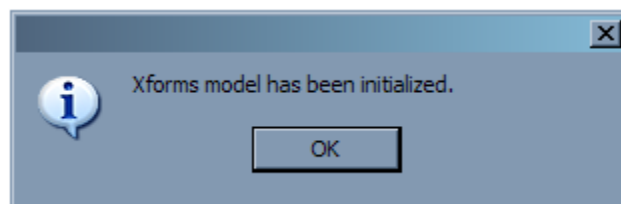


2. In the Properties View, choose **modal** from the **level** dropdown and **xforms-ready** from the **event** dropdown. The level setting is an XForms option which determines the style of the message box to be displayed. All messages are displayed as modal style message

boxes by the viewer, so it doesn't matter which setting is chosen here. To finish this event set the **Text** property to "The XForms model has been initialized".

[-] XForms	
Text	o The xforms model has
level	o modal
event	o xforms-ready
if	o xforms-ready
	o xforms-model-destru
	o xforms-bindings-exc
	o xforms-compute-exc
	o xforms-link-exceptio

- Test the event by viewing the form in the preview mode. The message should be displayed before the form is displayed.



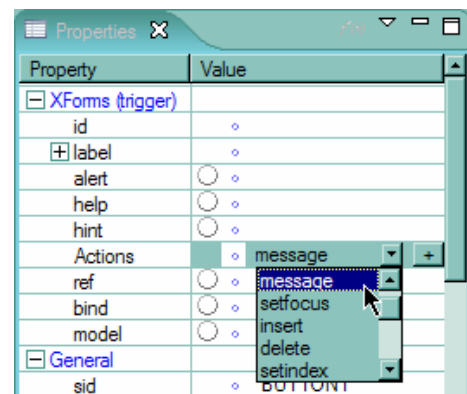
### Event 2 – DOMActivate

DOMActivate is the event that detects the activation of the presentation element that contains this event. For example, clicking a button registers the DOMActivate event for that button.

The code will look similar to this:

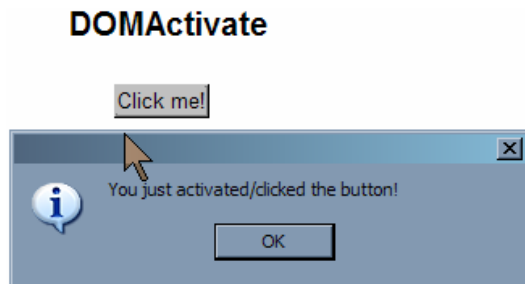
```
ev:event="DOMActivate"
```

- Select the Click me! Button.
- In the Properties View, expand **XForms > Actions** and add a **message** action. Select **message** from the drop down list and click the **+** button.
- Within the **message action**, set the text to "You just activated/clicked the button".
- Choose **modal** from the **level** drop down and set the event to **DOMActivate**.



[-] Actions	o 1 item
[-] mess...	o [X]
Text	o You just activated/clic
level	o modal
if	o
id	o
event	o DOMActivate

- Test the form in the preview mode. Click the button to fire the DOMActivate event.



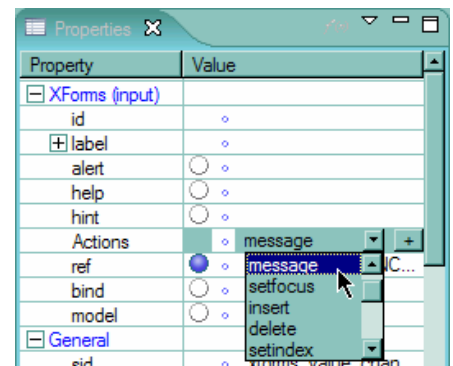
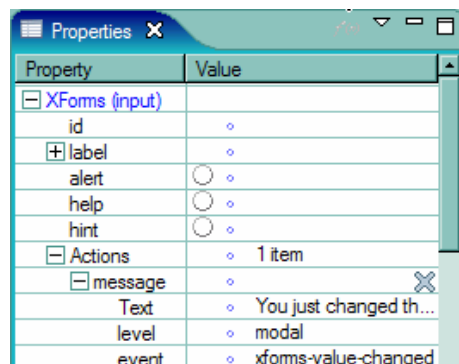
### Event 3 – xforms-value-changed

Occurs when a value is changed in an xforms:select or xforms:select1 option. In this part of the exercise we will create an action that is triggered when the user makes a choice from a popup.

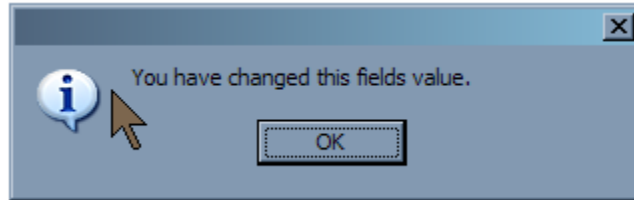
The code will look similar to this:

```
ev:event="xforms-value-changed"
```

- Select xforms\_value\_changed input field.
- In the Properties View, expand **XForms > Actions**. Select **message** from the drop down list and click the + button.
- Within the **message**, set the text to “You just changed the value”.
- Choose **modal** from the **level** drop down and set the event to **xforms-value-changed**.



5. Test the form in the preview mode. Change the value of the field and tab out to fire the **xforms-value-changed** event.



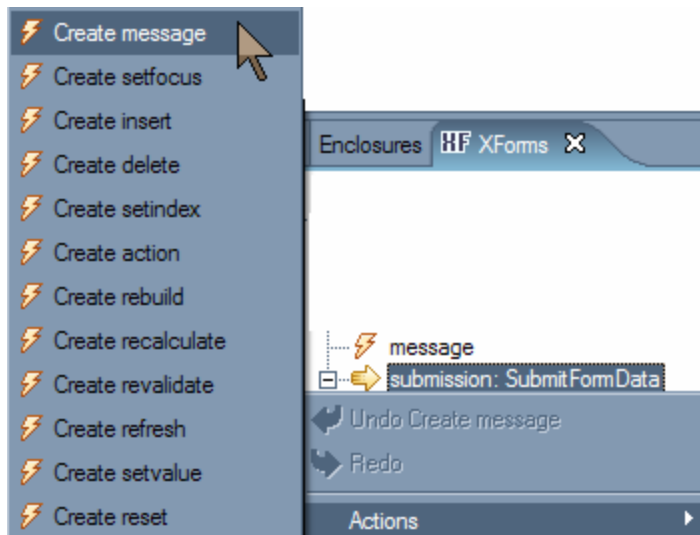
#### Event 4 – xforms-submit-done

Occurs when an XForms submission has successfully completed.

The code will look similar to this:

```
ev:event="xforms-submit-done"
```

1. The XForms submission button has been provided in the form with the submission rule. To make a submission event handler, right click on the Submission action in the XForms view and click on **Actions > Create message**.

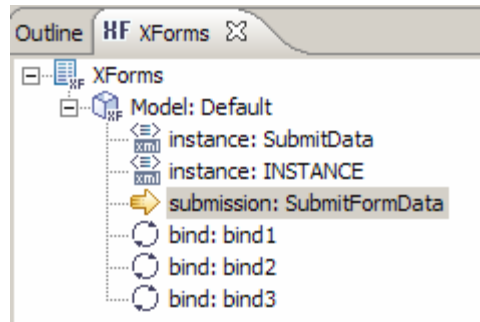


2. In the Properties View, enter **xforms-submit-done** as the **event**. Change the text to “The data instance has been successfully submitted to a file.”

[-] XForms	
Text	o The data instance has been
level	o modal
event	o xforms-submit-done



- The submission that is used needs a path where the submission will be sent. In the XForms view, select the SubmitFormData submission.

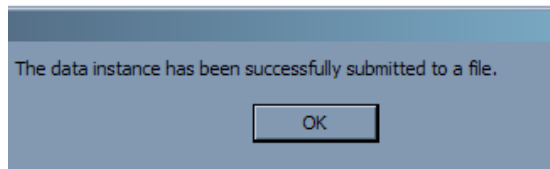


- The properties view now shows the properties for the submission. Change the action property to a valid path on your local machine.
- Test the form in the preview mode. Click on the **Submit** button to submit the instance and fire the **xforms-submit-done** event.

### xforms-submit-done

Submit

Occurs when an XForms submission has successfully completed.



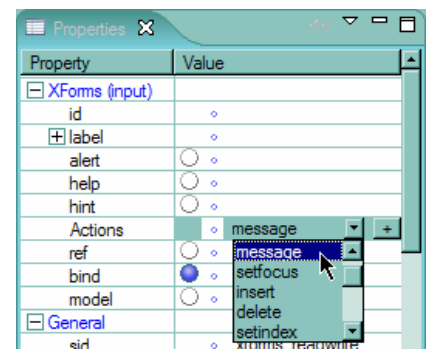
### Event 5 – xforms-required

This event occurs when a data node that is optional (required = false) becomes required, or when a data node that is required changes value and remains required. This event is triggered on the XForms control bound to that node. We will bind this event to an XForms input element, Spouse First Name. The form contains a radio group with three choices, 'Married', 'Single', and 'Divorced'. The input field becomes required when the user selects the 'Married' choice and optional when any of these other choices are selected.

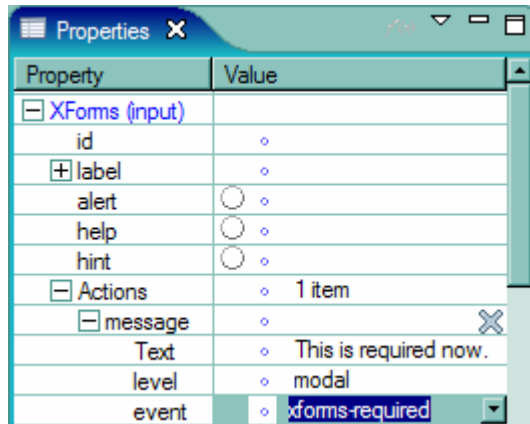
The code will look similar to this:

```
ev:event="xforms-required"
```

- You have been provided with the data model, binds and the XForms items. To create the event handler:



- Select `xforms_required` input field labeled “Spouse First Name”.
- In the Properties View, **XForms > Actions**. Select **message** from the drop down list and click the **+** button.
- Within the **message**, set the text to “This field is required now”.
- Choose **modal** from the **level** drop down and set the event to **xforms-required**.

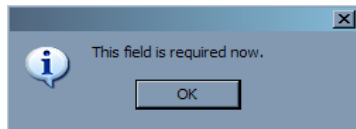


- Test the form in the preview mode. Click on the **Married** radio button to fire the **xforms-required** event.

#### xforms-required

Spouse First Name

Married  
 Single  
 Divorced



#### xforms-required

Spouse First Name

Married  
 Single  
 Divorced

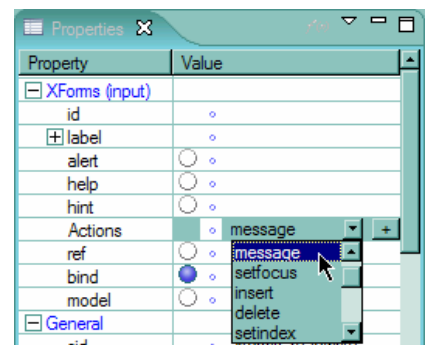
### Event 6 – xforms-readwrite

This event occurs when a data node that is read only (`readwrite = false`) becomes read-write, or when a node that is read-write changes value and remains read-write. This event is triggered on the XForms control bound to that node. The **Spouse First Name** field under **xforms-readwrite** label becomes writable when **Married** radio button is clicked and read only when any of the other radio options are checked.

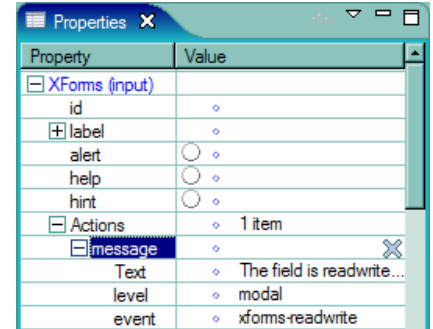
The code will look similar to this:

```
ev:event="xforms-readwrite"
```

- You have been provided with the data model, binds and the XForms items. To create the event.

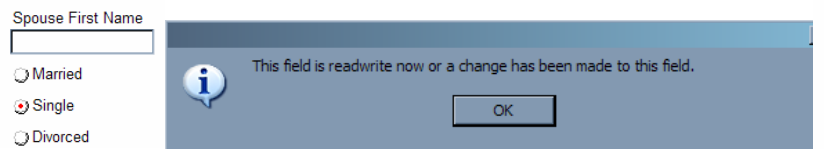


- a. Select `xforms_readwrite` input field labeled “Spouse First Name”.
- b. In the Properties View, **XForms > Action**. Select **message** from the drop down list and click the + button.
- c. Within the **message action**, set the text to “The field is readwrite now or a change has been made to the field.”.
- d. Choose **modal** from the **level** drop down and set the event to **xforms-readwrite**.

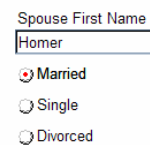


2. Test the form in the preview mode. Click on the **Married** radio button to make the field readwrite.

#### xforms-readwrite



#### xforms-readwrite



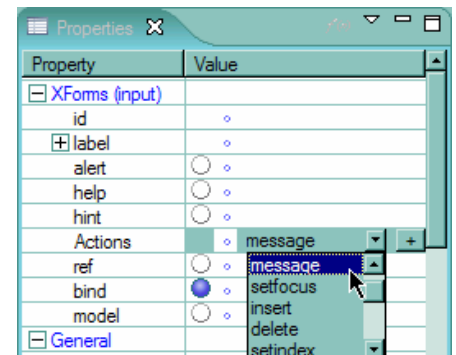
### Event 7 – xforms-enabled

This event occurs when a data node that is non-relevant (`relevant = false`) becomes relevant, or when a node that is relevant changes value and remains relevant. This event is triggered on the XForms control bound to that node.

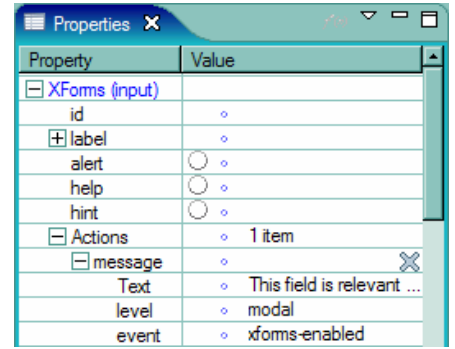
The code will look similar to this:

```
ev:event="xforms-enabled"
```

1. You have been provided with the data model, binds and the XForms items. To create the event.
  - a. Select `xforms_enabled` input field labeled “Spouse First Name” under the **xforms-enabled** label.



- b. In the Properties View, **XForms > Actions**. Select **message** from the drop down list and click the **+** button.
  - c. Within the **message action**, set the text to “This field is relevant now.”
  - d. Choose **modal** from the **level** drop down and set the event to **xforms-enabled**.
2. Test the form in the preview mode. Click on the **Married** radio button to make the field relevant.



**xforms-enabled**

- Married
- Single
- Divorced



**xforms-enabled**

- Spouse First Name
- Married
  - Single
  - Divorced

## 7.10. XForms Submissions

When submitting a form that contains an XForms data model, you can submit either the entire form or just a particular data instance. This makes it possible to send your data instance directly to processing applications rather than having to parse the complete form and extract the data instance.

If you want to submit a data instance, you must create a set of submission rules. These rules help determine what data is submitted, how the data is submitted, and where the data goes. In addition to submission rules, you must also create a submission button that is linked to the rules.

Each set of submission rules is inserted within the `<xforms:submission>` tag in the data model, as shown:

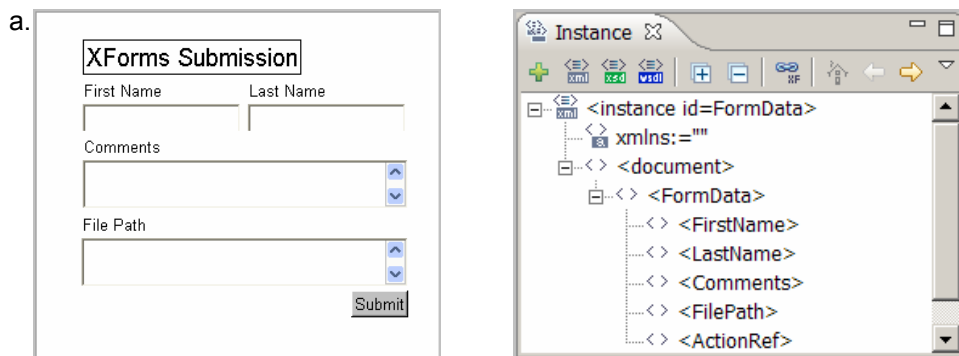
```
<xforms:model>
  <xforms:submission ...attributes...>
</xforms:submission>
</xforms:model>
```

Each submission is further defined by adding attributes to the `<submission>` tag and by including optional serialization rules. For additional information regarding creating and configuring an XForms submission refer to the XFDL Specification.

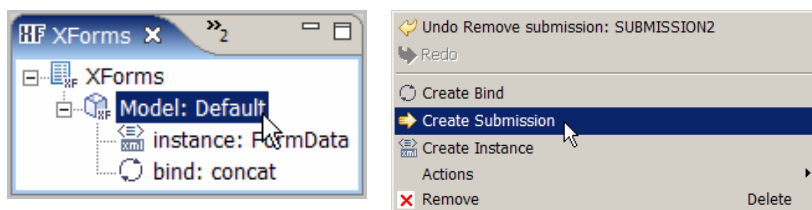
## 7.11. XForms Exercise 6 – Using XForms Submissions

This lesson is meant to show you how the Form Designer can help you create XForms submissions. In this example, you will be asked to create an XForms submission rule and configure a submit button to use it.

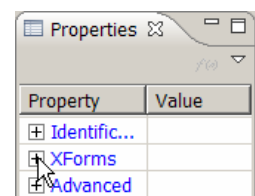
1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Switch to the Resource Perspective and open **XForms\_Ex06\_XFormsSubmission.xfdl** from the **TrainingExercises\_XForms** project in the Navigator View.
  - a. The Designer should automatically switch to the Designer Perspective and open the file with the instance as shown below.



3. In order to submit an instance in XForms you need to create a Submission rule that describes the characteristics of the submission. For this example follow these steps:
  - a. Right click on the Model: Default in the XForms view and select **Create Submission for Instance** (as shown below).



- b. Select the newly created submission item, open the **Properties View** (shown below) and expand the **XForms** heading. In this example, there are four values that we have to set for the submission rule.
  - i. The id of the submission,



Property	Value
id	SubmitFormData

- ii. The reference to the instance data that we will be submitting (ref)
  - iii. The submission method (method)
  - iv. The reference to the node in the instance that defines the submission location (actionref).
- c. Set the **id** attribute within the General heading to SubmitFormData

- d. The **ref** attribute contained in the XForms heading should contain an XPath statement that points to the instance that we want to submit. In this case it is the **FormData** instance.

Property	Value
ref	<input checked="" type="radio"/> instance('FormData')

- e. Since we are saving our instance locally we will choose **put** for the **method** attribute. Put serializes the data in the XForms model as XML. No response is expected (this is generally used with a file URL).

Property	Value
method	<input checked="" type="radio"/> put

Property	Value
replace	<input checked="" type="radio"/> instance

- f. For this example we must set the **replace** attribute to **instance** which replaces the instance specified by the instance attribute.
- g. A bind has been written that stores the filename and path in the ActionRef node of the FormData instance. The **actionref** attribute will have to contain an XPath statement that points to this location.


Property	Value
actionref	<input checked="" type="radio"/> instance('FormData')/FormData/ActionRef

---

Note: The action and actionref attributes both define the URL of the submission, however there are differences between them. The action property is a static string and cannot be computed. The actionref property was added to support XPath expressions so that the URL could be computed at runtime. The action is mandatory, but if actionref is present it will be used instead of the action.

---

- h. The action property is mandatory and therefore we must set it. Since the actionref also exists it will be used instead of the action, so it doesn't matter what value we give the action. Set the action property to "URL determined by actionref"
  - i. The submission rule is now complete.
4. The next step is to configure the submit button to use the submission rule you just created.
- a. Click on the submit button and open the Properties view for this item.
  - b. Find the **submission** attribute under the XForms heading and select **SubmitFormData**.

Property	Value
submission	 SubmitFormData

- c. The submit button is now configured.
5. Test your form by selecting the **Preview** tab.
  - a. Enter data into the **First name**, **Last name**, and **Comments** fields.
  - b. Since there are some permission restrictions on where the form can write the file, enter **c:\temp** in the **File Path** field.
  - c. Press the **Submit** button.
  - d. Finally, browse to c:\temp and look for a file called **FormData.xml** open up this file and verify that it looks similar to this:

```
<document xmlns=""
xmlns:custom="http://www.ibm.com/xmlns/prod/XFDL/Custom"
xmlns:designer="http://www.ibm.com/xmlns/prod/workplace/forms/designer/2.6" xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xfdl="http://www.ibm.com/xmlns/prod/XFDL/7.0"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <FormData>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
    <Comments>I love forms!!!</Comments>
    <FilePath>c:\temp</FilePath>
    <ActionRef>File:\\c:\temp\FormData.xml</ActionRef>
  </FormData>
</document>
```



## 7.12. XForms Tables

It is common for forms to have a repeating section. Prior to version 2.6, repeating sections were very tedious to implement because the repeating items had to be created and destroyed manually using XFDL functions.

XForms provides the ability to create a traditional table of repeated items organized into rows. This is accomplished by creating a template row that includes all of the items that should appear in each row. This row is then linked to the XForms data model. Each time a new row is added to the table, the template items are duplicated to create the new row. This occurs when the elements in the data model that are linked to those items are duplicated. This allows the table to expand to any size while ensuring that data for each row in the table is still maintained in the data model.

The Designer provides two ways to create a table. First, there is the manual process where the user selects the **Table (Repeat)** item from the palette which will lay down a table outline on the canvas. The empty table can then be populated with other items from the palette.

Alternatively, the Designer provides a table wizard (**Table (Repeat) by Wizard**) that walks the user through the steps to create a table. This option provides a Simple Setup and an Advanced Setup that uses existing data from an instance. Further customization provided through the table wizard includes setting table appearance, row editing, and row formatting options.

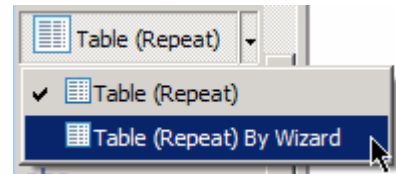
## 7.13. XForms Exercise 7 – Creating a Table (w/ Table Wizard)

This exercise demonstrates how to create a table with the Designer's Table Wizard.

### Exercise

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. "XForms\_Ex07\_TableWizard.xfd"), then click **Finish**
  - c. Click the Next Button and select the **Default Empty Form – XForms** Template. Click the Finish Button.
  - d. The Designer should automatically switch to the Designer Perspective and open the file.

3. Find the **Table** object on the XForms palette, change the active object to **Table (Repeat) By Wizard**.



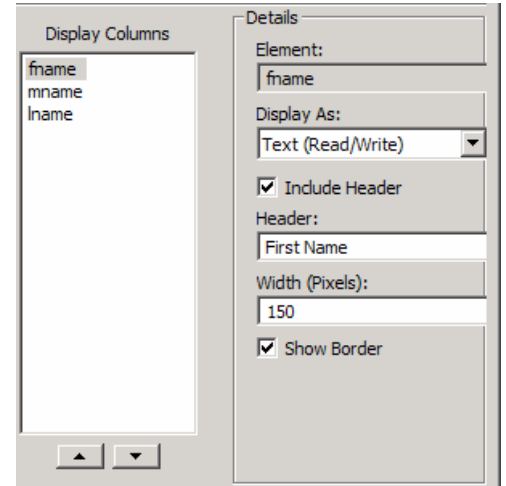
4. Click the palette object and then click on the canvas. This will initialize the Table Wizard.
5. Select **Simple Setup** and click **Next >**.
6. In the first section of the wizard we are going to **define the instance elements** that will be used to store the data contained within the table.

- a. Enter **fname** as a new element name and press **Enter**.
- b. Enter **mname** as a new element name and press **Enter**.
- c. Enter **lname** as a new element name and press **Enter**.
- d. Each element name will appear in the summary list on the left side of the dialog. The element order can be changed by selecting an element and then pressing the arrow buttons.
- e. Our table will start with one row, so leave the **Number of Initial Rows** field with the default value of 1.



- f. Clicking the **Advanced** button activates additional configuration settings. When you are creating a table you have the choice of creating a new instance (which is the default) or linking it to an existing instance. In this case we will just create a new instance.

- g. Click **Next >**
7. The second step is to **configure the columns**. For each element in the table we can define its read/write status, header and width.
- a. Select **fname** in the **Display Columns** listbox
    - i. Change the Header from “fname” to “First Name”
    - ii. Change the Width to 150
  - b. Select **mname** in the **Display Columns** listbox
    - i. Change the Header from “mname” to “Middle Name”
    - ii. Change the Width to 100
  - c. Select **lname** in the **Display Columns** listbox
    - i. Change the Header from “lname” to “Last Name”
    - ii. Change the Width to 150
  - d. Click **Next >**
8. Finally, we are going to customize some of the display setting of the table. We can change the table’s name, appearance, row editing settings and row formatting.
- a. Change the **table name** to “personTable”
  - b. Make sure that **Lines between Columns** and **Border around Table** are checked



**Table Wizard**

**Table Settings**  
Choose general display and configuration settings.

instance('INSTANCE')/table1/row

Table Name:

Table Appearance

Lines between Columns

Border around Table

Row Editing

Insert New Row Button

Remove Selected Row Button

Remove Button on Each Row

Row Formatting

Group Rows

Color Alternate Rows:  
First Row:  Second Row:

Highlight Row when Selected  
Highlight Color:

Margin  
Left:  Right:  Top:  Bottom:

< Back    Next >    Finish    Cancel

- c. Check the **Color Alternate Rows** checkbox. Select colors for the first and second rows by clicking on the color square to bring up the color dialog
- d. Click **Finish**. **Your table should resemble the one below.**

First Name	Middle Name	Last Name
<input type="text"/>	<input type="text"/>	<input type="text"/>

+ -

- e. Preview your form.

## 7.14. XForms Exercise 8 – Creating a Table (w/out Table Wizard)

The following is a step-by-step procedure for creating a table object. This table will support adding and removing rows.

Jane	C	Smith
John	J	Doe
Scott	L	Tiger

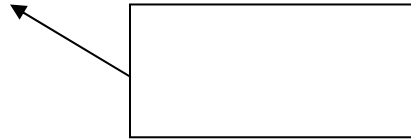
We are going to create a table that contains people's first, middle and last names.

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 > Designer**
  - b. If you created a project in a previous example and you want to reuse it skip step 2.
2. Create a new form file
  - a. In the file menu, select **File > New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. "XFormsTableEx"), then click **Finish**
  - c. The Designer should automatically switch to the Designer Perspective and open the file.
3. Create an instance
  - a. In the **Instance View**, click the button to create a blank xml instance.

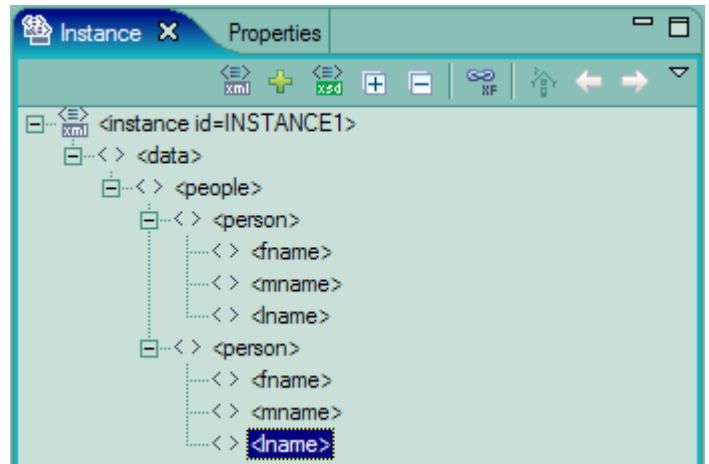


- b. Create the following instance:

```
<data>
  <people>
    <person>
      <fname></fname>
      <mname></mname>
      <lname></lname>
    </person>
    <person>
      <fname></fname>
      <mname></mname>
      <lname></lname>
    </person>
  </people>
</data>
```



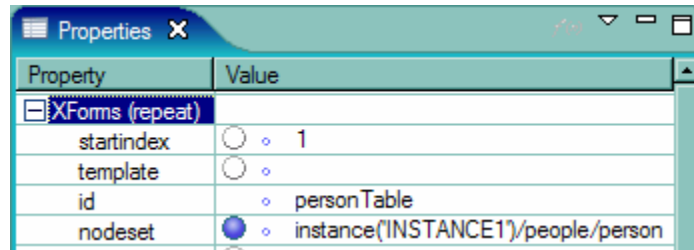
- c. Right-click the data element and select **Add Element**, a child element will be created. Rename it to “people”, by double clicking the new element and then typing the new name.
- d. Create the rest of the elements by following the same procedure.
- e. When you are finished your **Instance View** should look as shown to the right.



#### 4. Create a Table

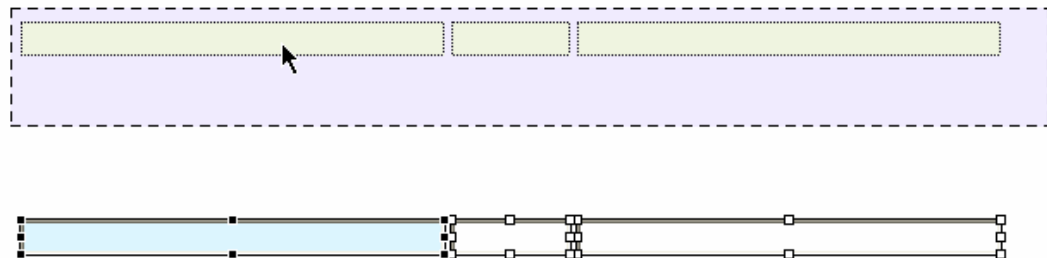
- Select the **Table (Repeat)** item from the palette and then click on the canvas. Manually resize the object so that it is large enough to place objects into. The table will resize to fit the items placed within it.
- We are going to give the table an id of “personTable”. Select the table and open the **Properties View**. Expand the “XForms (repeat)” property. Put “personTable” into the value of the “id” property.
- We are also going to bind the table to the “person” instance element. In the **Instance View** select the person element and drag it onto the table item. Select “update” from the dialog that appears.

- d. The Properties View should now look like the following:



5. Create the input fields

- Select the XForms Input (Field) item from the palette and then click on the canvas. Do this three times.
- Position the fields after one another so that they are along the same line and shrink the middle field. **\*\*Hint: use the after alignment modifier.\*\***
- Select all three fields and move them into the table item.



6. If you preview your form now you will notice that there are two rows visible. One is the template that will be used when adding additional rows. When we created our instance we created two persons, one of them will be the empty instance that is duplicated and the other is the first item in the table. We must make sure that the second occurrence is not visible. We will write a bind that will hide the template row.

- Open the **XForms View** and expand “XForms” and “Model: Default”
- Right-click “Model: Default” and select **Create Bind** from the menu
- Select the newly created bind in the XForms View and open the Properties View. Expand “General” and “Model Item Properties”.

Set the following properties:

**relevant = false()**  
*non-relevant nodes are omitted from XForms submissions. UI elements linked to this node (the three fields in this case) will not be displayed*

**nodeset** = people/person[position()=last()]  
*specifying that it is the last occurrence of person that we wish to hide*

Property	Value
[-] Identification	
id	o
[-] Model Item Properties	
type	<input type="radio"/> o
readonly	<input type="radio"/> o
required	<input type="radio"/> o
relevant	<input checked="" type="radio"/> o false()
calculate	<input type="radio"/> o
constraints	<input type="radio"/> o
[-] General	
nodeset	<input checked="" type="radio"/> o people/person[position()=last()]

Let's break this down:

- i. **people/person** is the path to the element that we want to bind
- ii. [ *equation* ] the equation in the square brackets is used to filter the nodes that are returned (that is, all person nodes are returned that meet the equation criteria).

**[position() = last()]** all person elements where their index (position) is not equal to the last one will be filtered out.

Now when you preview your form you should only see one row.

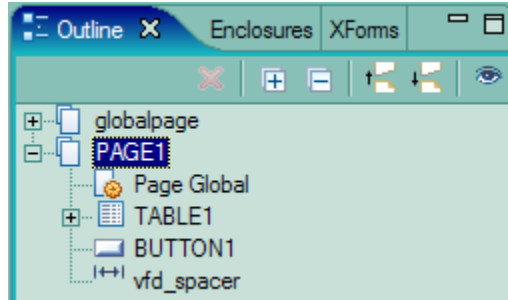
7. Create the Add Button
  - a. Select the **Trigger (Button)** item from the palette and click on the canvas.
  - b. Change the value to "Add"
  - c. Position the button below the table. It will be important to use relative placement because as items are added the table will grow. Select the Add button and then select the table (while holding down the shift key), then right-click one of the objects and select **Relative Align > Relatively Align Below**.

---

**Note:** If the Add button does not move then there may be an issue with your build order. If the button does not come after the table in the build order then it cannot refer to it. Your build order can be seen in the **Outline View** and should appear as shown:

---





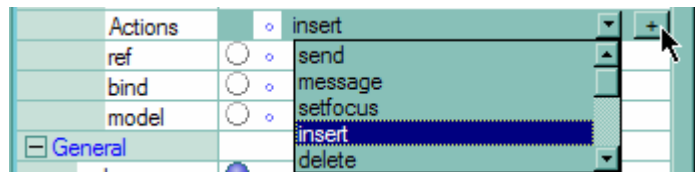
If you have to fix the build order then select the item that you want to move, click and hold the left mouse button and drag the item to its new location. A black line will appear indicating where the object will be placed within the build order; release the mouse button when the black line is where you want to position the selected item.

- d. Now it is time to add the XForms statements that “trigger” the addition of rows to this table. The two XForms statements we are going to add are **insert** and **setfocus**.

**i. insert**

Allows you to add a row of elements to a table. This function copies the last row of elements in the data model, then inserts the copy in the desired location in the data model. Once the copy is inserted in the data model, the table’s repeat creates corresponding items that are displayed to the user.

Select the Add button, open the **Properties View** and expand the XForms heading. Expand Actions/action/Actions and select insert from the drop down list and click the + button.



Expand “Actions” and “insert” to see the properties for this newly created XForms statement. We want to set the following properties:

**at** = index('personTable')

*an index number that sets the insertion point.*

**position** = after

*determines where the copy is placed, before or after the insertion point*

**nodeset** = people/person

*XPath to the collection of table rows in the data model where the last row is the row that will be duplicated.*

[-] Actions	o 1 item
[-] insert	o
at	o index(personTable)
position	o after
event	o DOMActivate
if	<input type="radio"/> o
nodeset	<input checked="" type="radio"/> o people/person
bind	<input type="radio"/> o
model	<input type="radio"/> o

## ii. setfocus

Setfocus will put the focus back on the table after the button has been clicked. Add this action in the same manner as insert.

**control** = personTable  
*an XPath reference to an element in the data model*

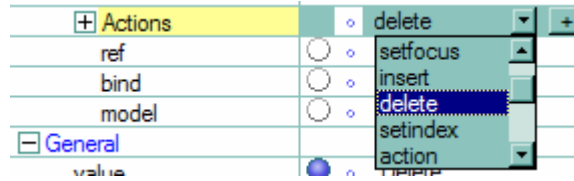
## 8. Create the Delete button

- a. Select the **Trigger (Button)** item from the palette and click on the canvas.
- b. Change the value to "Delete"
- c. Position the button after the "Add" button. It will be important to use relative placement because as items are added the table will grow.
- d. Select the "Delete" button first then the "Add" button, right-click one of the selected items and select **Relative Align > Relatively Align After**. This will position the "Delete" button directly after the "Add" button, using relative positioning such that when the "Add" button moves so will the "Delete" button.
- e. Now it is time to add the XForms statements that "trigger" the removal of rows from this table. The two XForms statements we will add are **delete** and **setfocus**.

### i. delete

Deletes a row of elements from a table. The elements are first deleted from the XForms model, then the table's repeat deletes the visible items that were linked to those data elements.

Select the button, open the **Properties View** and expand the XForms heading. Expand Actions/action/Actions and select delete from the drop down list and click the + button.



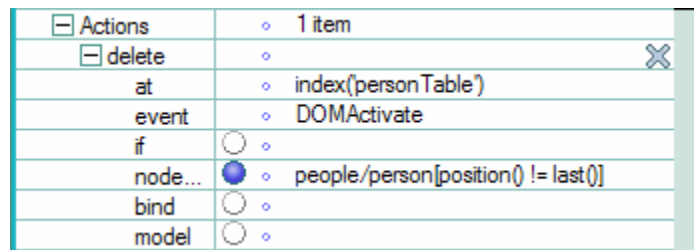
Expand “Actions” and “delete” to see the properties for this newly created XForms statement. We want to set the following properties:

**at** = index('personTable')

*an index number that determines which row to delete*

**nodeset** = people/person[position() != last()]

*specifying that it is the last occurrence of person that we wish to hide*



## ii. setfocus

Setfocus will put the focus back on the table after the button has been clicked. Add this action in the same manner as delete.

**control** = personTable

*an XPath reference to an element in the data model*

## 9. Bind the table and fields to the instance

- Binding instance data to input fields is very simple. Open the **Instance View** and expand each layer so that you can see the elements frame, mname and lname.
- Select the frame element, click and hold the left mouse button and drag the element over top of the first field. Repeat this for the mname and lname for their corresponding fields.
- To confirm that the binds have been created properly you should see the following in the Properties View for each field:

The image shows a screenshot of the 'Instance Properties' dialog in an XForms editor. The dialog has two tabs: 'Instance' and 'Properties'. The 'Properties' tab is active, displaying a table with 'Property' and 'Value' columns. The 'XForms' tree is expanded to show an 'input' element. The 'ref' property of this 'input' is highlighted in yellow and set to 'fname'. Below the dialog, two other 'ref' properties are shown, one for 'mname' and one for 'lname', both also highlighted in yellow. Arrows point from a text box to these three 'ref' properties.

Property	Value
input	
inremen...	<input type="checkbox"/> false
id	
label	
alert	
help	
hint	
Actions	<empty>
ref	fname

Each field is bound to the matching instance element

## 7.15. XForms Pane

Sometimes electronic forms employ techniques that are not used on traditional paper forms. An example is a form section that contains multiple overlapping sections whose visibility is controlled by another input object. In previous versions of the Designer this was a very tedious process because the XFDL language did not contain a mechanism for grouping objects together. The visibility for each object had to be controlled independently, which meant creating a compute in every object.

The 2.6 product introduces the Pane object which allows the grouping of objects together. If the pane is invisible then all the items that are contained within it are also invisible. With the introduction of the pane we can also leverage some XForms UI Controls; `xforms:switch` and `xforms:case`. An `xforms:case` is represented by a XFDL pane. The `xforms:switch` can contain multiple `xforms:cases` and allows us to define which `xforms:case` is visible.

## 7.16. XForms Exercise 9 – Create a Multi-Pane Section

The following is a step-by-step procedure for creating a multi-pane object in the Workplace Forms Designer 2.6.

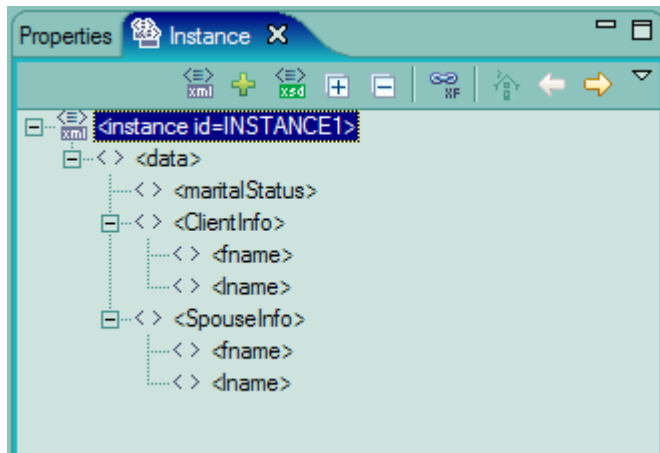
1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
  - b. If you created a project in a previous example and you want to reuse it skip step 2.
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. “XFormsPaneEx”), then click **Finish**
  - c. Click the Next Button and select the **Default Empty Form – XForms** Template. Click the Finish Button.
  - d. The Designer should automatically switch to the Designer Perspective and open the file.
3. Create the following instance

```
<data>
  <maritalStatus></maritalStatus>
  <ClientInfo>
    <fname></fname>
    <lname></lname>
  </ClientInfo>
  <SpouseInfo>
    <fname></fname>
    <lname></lname>
  </SpouseInfo>
</data>
```



- a. In the **Instance View**, click the button to create a blank xml instance.
- b. Right-click the data element and select **Add Element**, a child element will be created. Rename it to “*maritalStatus*”; double click the new element (to edit) and then type in the new name.
- c. Create the rest of the elements by following the same procedure.

When you are finished your **Instance View** should look like the following:



4. Create a radiogroup that will control which case is displayed.



- a. Select the **RadioGroup (Select 1)** item from the Standard Library on the palette and click on the Design canvas.
- b. Set the label of the radiogroup to “Marital Status”, this property is beneath the “General” category.
- c. Add 2 items to the radio group (Single, Married)



- i. Select the **Choice (Item)** from the Standard Library on the Palette and click the radiogroup item on the canvas. Do this a second time to create a second radio.
- ii. Select the radiogroup item on the canvas and open the **Properties View**
- iii. Expand **XForms (select1)**
- iv. Expand Item > item > value and enter “single” as its text value, then expand label and type “Single” as its text value.
- v. Expand the second item and set its value to “married” and label to “Married”.

[-] XForms (select1)	
selection	<input type="radio"/> closed
itemset	<input type="radio"/>
[-] Item	<input type="radio"/> 2 items
[-] item	<input type="radio"/> [down] [X]
[-] value	<input type="radio"/>
Text	<input type="radio"/> single
[+] extension	<input type="radio"/>
[-] label	<input type="radio"/>
Text	<input type="radio"/> Single
Actions	<input type="radio"/> <empty>
[-] item	<input type="radio"/> [up] [X]
[-] value	<input type="radio"/>
Text	<input type="radio"/> married
[+] extension	<input type="radio"/>
[-] label	<input type="radio"/>
Text	<input type="radio"/> Married
Actions	<input type="radio"/> <empty>

d. Position the “Married” radio so that it follows the “Single” one.

- i. Select the “Married” radio and then the “Single” Radio, right-click and choose **Relative Align > Relatively Align After Previous Item**

- e. Bind the radiogroup to the **maritalStatus** element. Select the element in the **Instance View**, left-click and hold the mouse button and drag it over the radiogroup, then release the mouse button to apply the link. This will set the **ref** property to “instance('INSTANCE1')/maritalStatus”

- 5. Create the first pane object that will control the switching.



- a. Select the **Pane (Switch)** item from the **XForms palette** (if you don't see it, look for **Pane (Group)** on the palette, the **Pane (Switch)** is located in the popup menu that defaults to **Pane (Group)**), click and drag it onto the canvas so that it is large enough to place other objects inside it. By default the first Case is created when a Switch is created.
- b. Select the newly created object (the sid is “PANE1”) and open the **Properties View** to change the background color. Expand “Appearance” and locate the **bgcolor** property, choose a color.

- 6. Setup Case 1 – The client information



- a. Create a pane that will be used to group all the items together. Select the **Group** item from the palette and add it to the XForms Switch item.
- b. Expand the inner pane (group) so it will be large enough to hold the two fields and label.
- c. Select the inner pane and open the **Properties View** to change the background color. Expand “Appearance” and locate the **bgcolor** property, choose a color.

The screenshot shows a form with a purple header labeled "Client Information". Below the header are two input fields: "First Name" and "Last Name". The form is set against a light green background.



- d. Add a **Label (output)**
  - i. Use the Properties View to change the label text to match that in the screenshot.



- e. Add two **Field (input)** items to the case. Label text can be added to the fields through the XForms (input) heading in the Properties view.
- f. Bind the two fields on this case to the corresponding elements of “ClientInfo” in the instance.
  - i. Open the **Instance View**, expand the entire tree.
  - ii. Select fname, click and hold the left mouse button and drag it overtop of the first name field then release the mouse button.
  - iii. Select lname, click and hold the left mouse button and drag it overtop of the last name field then release the mouse button.

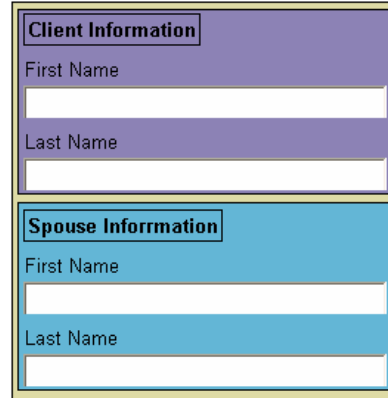


7. Create and Setup Case 2 – The client and spouse information



- a. Select the **Case** item from the **Standard Library** on the palette, click on the Design canvas inside of the XForms Switch item (the outer pane).

When you add a case to a Switch item it automatically becomes the active case. To modify any of the other cases you must first make it the active case. This can be accomplished by selecting the Switch (outer) pane, right-click and choose **Active Case > caseX**. Only the active case and the items it contains will be visible.



- b. Create two panes (the **Group** item in the Palette) that will be used to group all the items together. The first will be for Client Info (exactly the same as the one on the first case) and the second will be for Spouse Info.
- c. Resize the panes so that they match the screen shot.
- d. Select each inner pane and open the **Properties View** to change their background color. Expand “Appearance” and locate the **bgcolor** property, choose a color.



- e. Add a **Label (Output)** to each group.
  - i. Change the label text to match that in the screenshot. Use the **Properties View**.



- f. Add two **Field (Input)** items to each case.
- g. Bind the two fields in the client pane to the corresponding elements of “ClientInfo” in the Instance and the two fields in the spouse pane to the corresponding elements of “SpouseInfo” in the Instance.
  - i. Open the **Instance View**, expand the entire tree.
  - ii. Select fname, click and hold the left mouse button and drag it overtop of the first name field then release the mouse button.
  - iii. Select lname, click and hold the left mouse button and drag it overtop of the last name field then release the mouse button.

8. Add the Switching Functionality

- a. Select the radiogroup, open the **Properties View** and expand the **XForms** category.

[-] Actions	o 1 item
[-] action	o
[-] Actions	o
[-] toggle	o
case	o CASE1
if	o . = 'single'
[-] toggle	o
case	o CASE2
if	o . = 'married'
id	o
event	o xforms-value-changed
if	o

- b. Under **Action**, select **action** from the popup and click the add button.
  - c. Expand the item that was just created.
  - d. Under **Actions**, add two **toggle** actions. Select toggle from the popup and click the add button. Do this twice.
  - e. Expand the items that were just created. Now we must configure the two toggle actions that we added in the previous step.
    - i. The first toggle will point to “CASE1” (set case property to **CASE1**) if the “single” radio is selected (set if property to . = ‘**single**’)
    - ii. The second toggle will point to “CASE2” (set case property to **CASE2**) if the “married” radio is selected (set if property to . = ‘**married**’).
9. Open the form in the Preview to confirm that the panes switch when the radio button is toggled from “single” to “married”.

## 7.17. Web Services and Schemas

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix applications.

An XML Schema is a language for describing the structure and constraining the contents of XML documents. A schema guarantees that the XML elements linked to it follow the defined structure

The purpose of an XML Schema is to define the legal building blocks of an XML document

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

## 7.18. XForms Exercise 10 – Implementing Web Services in XForms

This exercise demonstrates how to implement a Web Service within a form. This assumes that you already have a WSDL that can be used as the starting point.



### Prerequisites

- This exercise requires that the student have access to the Internet.
- This exercise is dependent on a Web Service that is not hosted by IBM and therefore its availability can not be guaranteed. The service used in this exercise can be substituted for any other valid Web Service.

### Exercise

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. “XForms\_Ex10\_Web Services.xfdl”), then click **Next**
  - c. Select the **Default Empty Form – XForms** Template and click **Finish**.
  - d. The Designer will automatically switch to the Designer Perspective and open a blank form.
3. First, we will enclose the Web Service WSDL called **XForms\_Ex10\_spellcheck.wsdl** that is included in the **TrainingExercises\_XForms** project.
  - a. Open the **Enclosures View** and expand **WSDL**.
  - b. Right-click on **Web Services** and select **Enclose WSDL File**.
  - c. In the new dialog that appears, browse to the location of the **XForms\_Ex10\_spellcheck.wsdl** file and click **Open**.
  - d. In the **Enclosures View** if you expand **Web Services** you should now see that the WSDL has been added to the form.
4. Add 3x **Field (Input)** and a **button (Submit)** to the Designer canvas.
  - a. Position and layout are not crucial here. Feel free to “dress up” the form as you like.
  - b. The first **Field (Input)** will contain the word you want to spell check.
    - i. In the Properties view, change **XForms (Input) > label > Text** to ‘Enter the word to spell check’



- c. The second **Field (Input)** will display the result of the Web Service.
    - i. Change the **XForms (input) > label > Text** property to 'Web Service Result'
  - d. The third **Field (Input)** will hold the ID key we need for this Web Service.
    - i. Change the **XForms (input) > label > Text** property to 'ID Key'
  -  e. The **Submit (button)** will trigger the call to the Web Service.
    - i. Change the **General > value** property of the button to 'Spell Check'.
5. XForms Instances are the foundation of all XForms forms. The instance structure for this form will be generated from the WSDL that we embedded in the form in a previous step.
- a. Open the **Instance View**
  -  b. Click the button that generates an instance from the enclosed WSDL
  - c. Click the checkbox for both messages: **doSpellingSuggestion** and **doSpellingSuggestionResponse**
  - d. Click **OK**
6. Now that our items have been created we must link them to an XForms Instance.
- a. In the Instance View, expand both instances that were created from the previous step
  - b. In the first instance, select the **<phrase>** element, hold the left-mouse button, drag it over the '**Enter the word...**' field and release the left-mouse button. This will create a "link" between the form item and the XForms instance element.
  - c. In the first instance, select the **<key>** element, hold the left-mouse button, drag it over the '**ID Key**' field and release the left-mouse button. This will create a "link" between the form item and the XForms instance element.
  - d. In the second instance, select the **<return>** element, hold the left-mouse button, drag it over the '**Web Services Result**' field and release the left-mouse button. This will create a "link" between the form item and the XForms instance element.
7. When the button is pressed we want to call the web service and place the result into the '**Web Services Result**' field. (To accomplish this we will link the button to an XForms Submission).
- a. Open the **XForms View**, expand **XForms > Model:Default**
  - b. Right-click **Model:Default** and select **Create Submission**
  - c. Select the submission in the **XForms View**, and open the **Properties View**.
  - d. Expand the **XForms** category. We need to set the **ref** property to the instance that contains the request.

- i. Beside **ref**, type: instance('INSTANCE')
- e. Now we need to set the **instance** property to the instance that will store the result.
  - i. Beside **instance**, type: INSTANCE1
- f. We need to set the **action** property to the URL of the web service. The URL can be seen as part of the WSDL
  - i. The URL is **http://api.google.com/search/beta2**
  - ii. Copy this URL into the **action** property of the XForms Submission

---

Note: To find this URL, click the **Source** tab to open the Source View. Look for the element called **soap:address**. This element has an attribute called **location**

---

- g. Set the **method** property to **post**.
- h. Find the **mediatype** property and select **application/soap+xml**
  - i. Open up the **Source** tab and find the **xforms:submission** entry.
  - ii. Change the **mediatype** attribute to look like this **application/soap+xml; action=urn:GoogleSearchAction**

---

Note: To find this value, click the **Source** tab to open the Source View. Look for the element called **soap:operation**. This element has an attribute called **soapAction**. It is placed in the header of the submitted request and is used to aid the web service in determining which method has been called.


---

- i. Return to the design view by clicking on the **Design** tab. Return to the properties view for the submission. Expand the **XForms** category. Find the **replace** property and select **instance** from the drop down box.
  - j. Select the **button** on the canvas, and open the **Properties View**
  - k. Expand the **XForms (submit)** category and set the submission property to **SUBMISSION** (it will be the only item in the popup).
- 8. Now preview the form to test the Web Service. Keep in mind that you must have access to the internet and this Web Service must be active. **This service is not maintained by IBM and is therefore not guaranteed to work at all times.**
  - a. Enter the following Key in the ID Key field: 1tnBit5QFhLMq2jrDsaTUtk40yw/s6Fy
  - b. Type 'allows' in the field
  - c. Click the 'Spell Check' button.
  - d. The 'Web Service Result' field now shows 'allowed'.

## 7.19. XForms Exercise 11 – Implementing Schema Validation

This exercise demonstrates how to implement a form that uses a schema for client-side validation.

### Exercise

1. Open Workplace Forms Designer 2.6
  - a. **Start > Program Files > Workplace Forms Designer 2.6 – Designer**
2. Create a new form file
  - a. Select your project, right-click and select **New > New Workplace Form**
  - b. Select your project as the parent folder and enter a filename (i.e. “XForms\_Ex11\_SchemaValidation.xfd”), then click **Finish**
  - c. Click the Next Button and select the **Default Empty Form – XForms** Template. Click the Finish Button.
  - d. The Designer should automatically switch to the Designer Perspective and open the file.
3. Enclose the XML Schema file called XForms\_Ex11\_schema.xml that is included in the TrainingExercises\_XForms project.
  - a. Open the **Enclosures View** and expand **Schema**.
  - b. Right-click on **Model:Default** and select **Enclose Schema File**.
  - c. In the new dialog that appears, browse to the location of the **XForms\_Ex11\_schema.xml** file and click **Open**.
  - d. In the **Enclosures View**, under **Schema** and the default model you should see that the schema has been added to the form.
4. Now we will create an XML instance, which will store all the form data. The schema will validate the structure and content of this instance. This instance can be generated automatically.
  - a.  Open the **Instance View** and click the button that generates an instance from an embedded schema file. Since there is only one schema embedded in the form it is used by default, you would be given the choice if more than one existed.
5. Create the input items that are linked to the schema-generated instance.
  - a. In the **Instance View**, expand the data and person elements. You should now see the entire structure of the XML Instance.

- b. With the mouse, select the **Person** element. Click and hold the left mouse button, drag the Person element onto the canvas and release the mouse button. This will create a pane that contains a field for every child element contained within the Person element. Each field is already linked through XForms ref attributes.
6. Preview the form to test the schema validation.
  - a. Click the **Preview** tab at the bottom left corner of the designer canvas.
  - b. The schema defines six data elements:
    - i. First Name –Has a maximum length of 10
    - ii. Last Name – Has a maximum length of 10
    - iii. Email Address – Has a maximum length of 35
    - iv. Phone Number – The string must match the pattern `(###) ###-####`
    - v. Age – The integer must be  $\geq 55$
    - vi. Birth Date – Must have the format YYYY-MM-DD and cannot be empty.
  - c. Enter data into each of the fields, if it does not adhere to the above rules then the field will be flagged invalid.





## 8. Choosing Between XFDL and XForms

### 8.1. Audience

This section is intended to describe some of the differences between XFDL and XForms. In form development each has its place and it is important to learn how and when each method should be leveraged.

### 8.2. Overview

Both XForms and XFDL have event-driven compute engines that can be used to manipulate a form. The XFDL language has been modified to serve as the presentation layer or “skin” off the underlying XForms model, however there are still several areas where these two implementations overlap each other. While designing a form you may be faced with several situations where you are unsure of which method is most appropriate, XFDL or XForms. A simple rule to follow is: **Any logic or behavioral effects that are dependant on business rules should be controlled by XForms, where possible.** Because XForms is still in its infancy there will be times where the language does not contain the necessary constructs required to produce a desired result. In these cases it is appropriate to use the functionality of XFDL to supplement the deficiencies in XForms. Using XForms over XFDL procedures is also important if you are interested in exporting your XForms model. Any logic that is defined using XForms binds will be included in the export procedure, which makes it much easier to implement the XForms model with a different presentation layer (i.e. XHTML).

### 8.3. Examples Implementing XForms vs. XFDL:

#### 8.3.1. Item Visibility

Create an XForms bind with the relevant attribute instead of placing a compute on the visible option of an item. The following example is common when hiding a delete row button if there is only one row in the table.

```
<visible compute="custom:rowCount > 1 ? 'on' : 'off'" />

<bind nodeset="rowCount" relevant=" . > 1" />
```

The visible option is a child of the button being hidden. The XForms bind uses an element in the instance data, to which the delete button is bound (using “xforms:trigger” where the ref attribute points to the same “rowCount” element), to determine whether or not it should be shown. The advantage of the XForms method is that the relevancy not only affects the visibility on the screen, but if the “rowCount” element is not relevant then it will not be submitted during an XForms submission.

#### 8.3.2. Item Status: Mandatory or Optional

Create an XForms bind with the **required** attribute instead of placing a compute on the format option to determine a field’s mandatory status. The following example is a common implementation of a field that is mandatory when the user selects the “other” checkbox, which requires the user to specify the item explicitly.

```
<format>
  <constraints>
    <mandatory compute="CHECK1.value" />
  </constraints>
</format>

<bind nodeset="otherField" required=" ../otherCheck = 'true'" />
```

The first code snippet is an example of a compute within the format option that toggles the mandatory status of the field based on the value of the check box.

The XForms bind uses two elements in the instance data. The first is “otherField” which will store the result of the bind; if the check is on we want the required attribute of the bind to be “true”, otherwise “false”. The instance element “otherCheck” stores the value of the checkbox, either “true” or “false”. When you link the “other” field to this bind the result is a field that becomes mandatory when a check box is on.

### 8.3.3. Item Value: Performing Simple Calculations

Create an XForms bind with the **calculate** attribute instead of placing a compute on the value option to dynamically calculate a field’s value. The following example is a common implementation of a field whose value is the result of a mathematical calculation.

```
<value compute="priceField.value * '1.07'"></value>

<bind nodeset="totalPrice" calculate="../price * 1.07" />
```

The first code snippet is an example of a compute within the value option that gets set to the result of the priceField multiplied by “1.07”.

The XForms bind uses two elements in the instance data. The first is “totalPrice” which will store the result of the bind; the product of “price” and “1.07”. The instance element “price” contains the current price as a float. When you link the “total” field to this bind its value will contain the product stored in the “totalPrice” element.

## 9. Appendix A - Learning Eclipse Components

The new Workplace Forms Designer has been built on the Eclipse platform. By building it on an existing framework we have been able to leverage several features that are already available without having to create them.

Learning the Eclipse platform can be a bit daunting at first but there are many form building techniques that are better suited to this new paradigm. This section has been written to help prepare you for the task of building forms in the new Designer. The Eclipse components will be discussed in the context of how IBM's Workplace Forms Designer leverages them to build XFDL forms.

### 9.1. What is a Perspective?

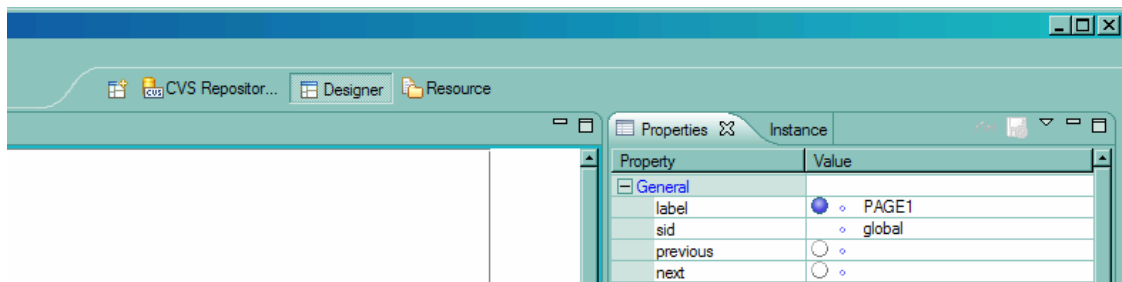
Eclipse is composed of perspectives and views. A perspective is used to define what views are present and where they appear on the screen. A view is a moveable window that provides a specific functionality. Some examples of views are the properties view (which displays all the properties of the selected item), design view (drawing space), and outline view (shows all the objects that have been created). Each of these views can be added to a perspective so that every time the perspective is opened these views will be present. Multiple perspectives can share the same views.

The Workplace Forms Designer comes with two default perspectives. The Designer perspective is the default perspective for creating and modifying forms. The Views that belong to the Designer perspective are the: properties view, instance view, design view, source view, preview, the item palette, the outline view, enclosures view, problems view, tasks view, bookmarks view and XForms view. For additional information about Perspectives please refer to the Eclipse help, which can be found by selecting **Help > Help Contents** from the **File** menu and entering "perspectives" as the search criteria.

#### 9.1.1. How Do I Activate a Perspective?

Eclipse comes with several other perspectives. Perspectives can be activated from the File Menu; select **Window > Open Perspective > Other...** A dialog will appear that contains all of the perspectives that are available. Select the perspective you wish to open and click **OK**.

The perspectives will appear as tabs in the upper right-hand corner of the interface. Each tab is a button linked to its perspective, pressing the Designer button activates the Designer perspective.



### **9.1.2. Customizing a Perspective**

The layout of the views within a perspective can be customized to suit your needs. All of the Views are designed as “tabs” within the Designer interface, if you click and hold a tab then you can move it to a different location in the Perspective. The View will “snap” into place as you move it around the perspective. Take some time once you have familiarized yourself with the multiple views to customize their arrangement.

### **9.1.3. Resetting a Perspective**

If you have customized the layout of a perspective and want to revert back to the original layout, right-click the perspective button (in the top right-hand corner) and select Reset.

### **9.1.4. Closing a Perspective**

Closing a perspective will remove the corresponding button from the perspective toolbar. Close a perspective by right-clicking its button and select Close. Hidden or closed perspectives can be re-opened from the File Menu.

### **9.1.5. Saving a Customized Perspective**

Perspectives will automatically “remember” any changes made to their layout. However if you want to customize a default perspective and save it with a different name then right-click the perspective button and select Save As... Once you give your perspective a name then a button will be created in the perspective tab for quick access to it.

## 9.2. The Resource Perspective

The Resource perspective is the default perspective for managing projects and files. The Resource perspective contains all the previous views but adds the Navigator view and its layout is also different from the Designer perspective.

### 9.2.1. Managing Your Files

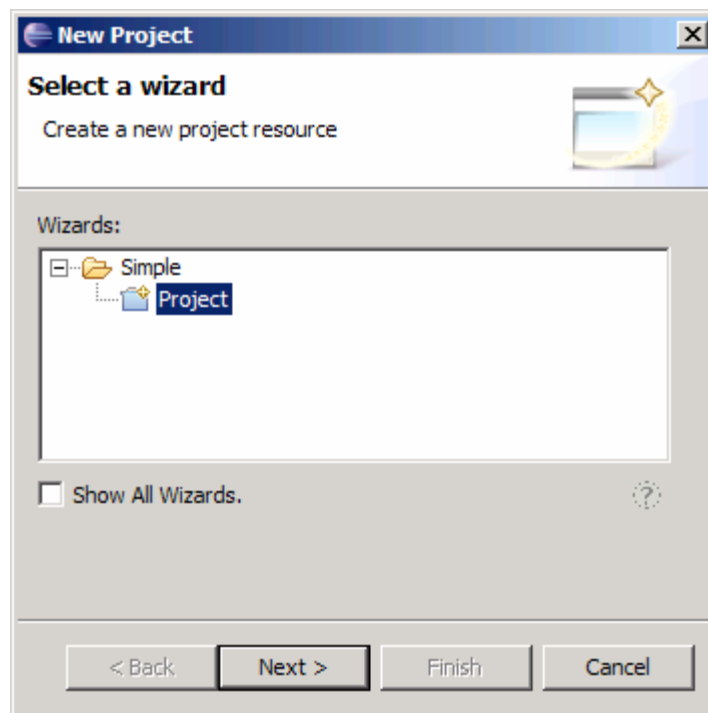
#### What is a Project?

A project is the basis for all form development; it groups all the files related to a particular function or application. All forms must exist within a project.

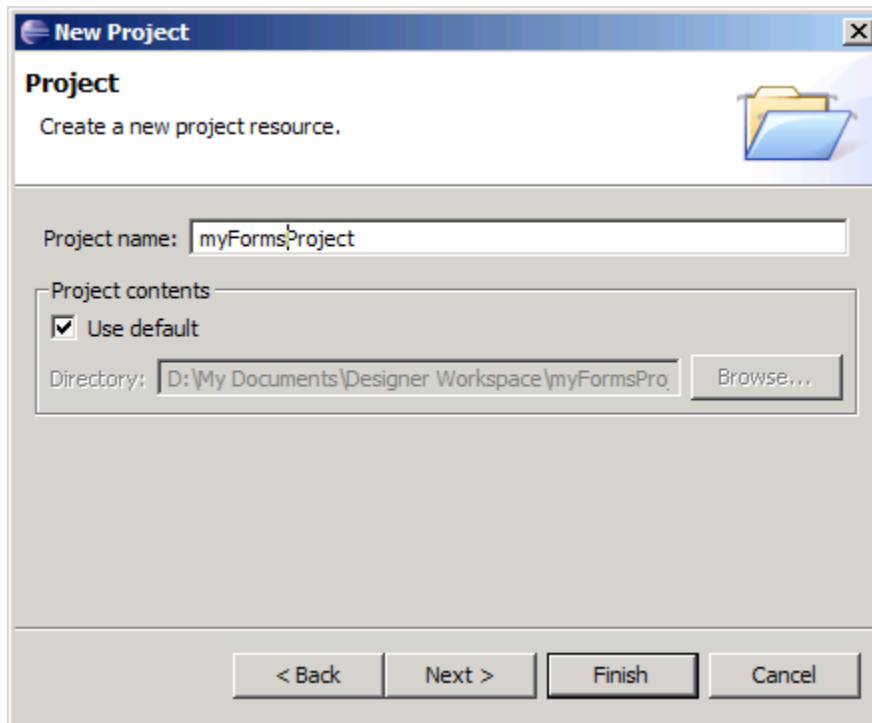
#### How Do I Create a Project?

A project can be created from the **File** Menu:

1. Select **File > New > Project**.
2. Select **Next**.



3. Enter a project name. By default the project will be created in the Eclipse Workspace (that is can be defined every time Eclipse is launched).

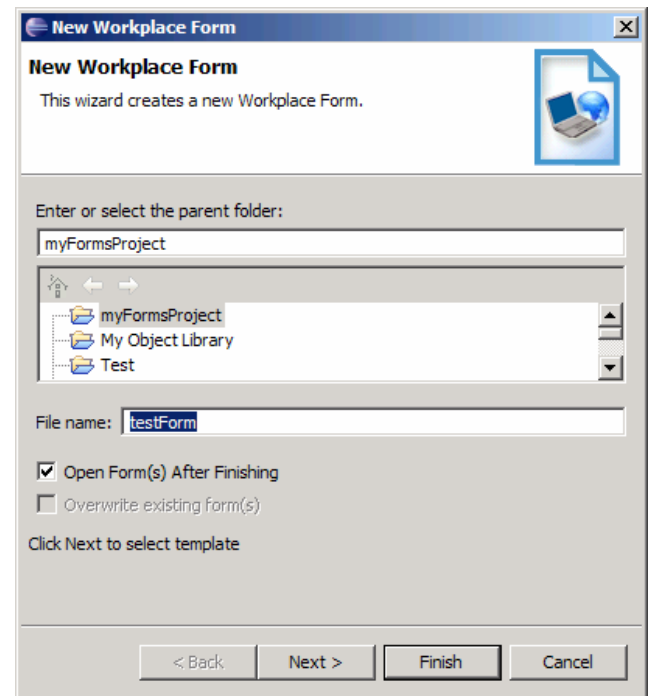


4. Click **Finish**.

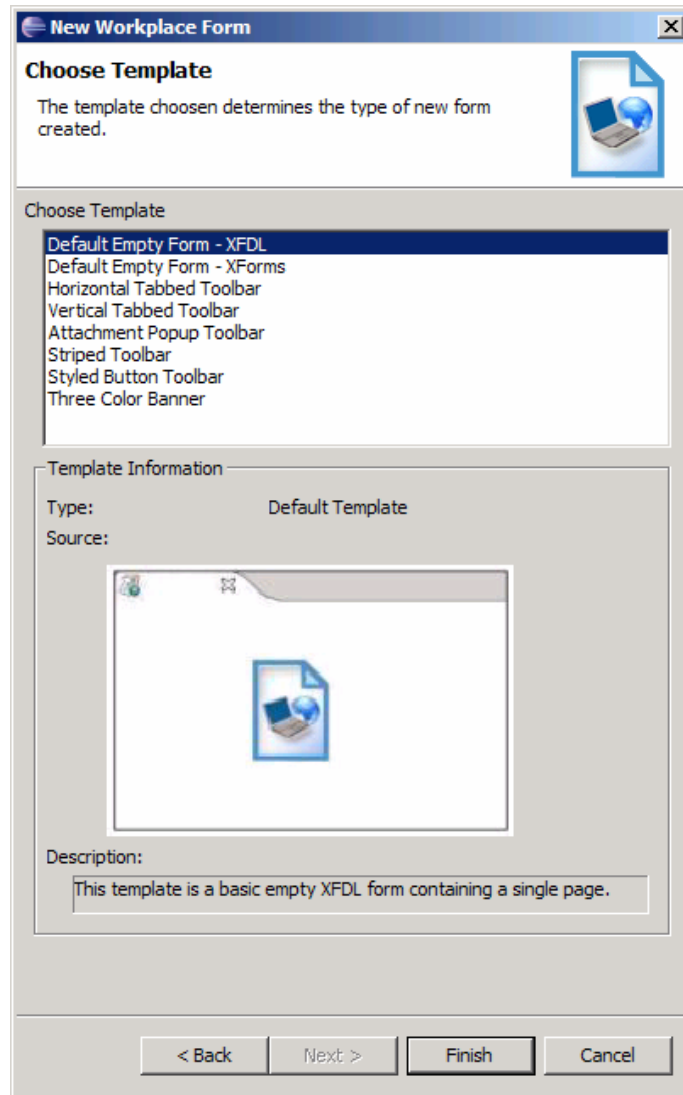
#### How Do I Create a New File?

Once the project has been created then it can be seen in the **Navigator View**. Open the Navigator View and look for your new project. Right click the project and select **New > New Workplace Form**.

Provide the project where the file will be stored and the name of the file.



Click **Next >**. Now you can choose between a series of templates. Select the desired template and click **Finish**.



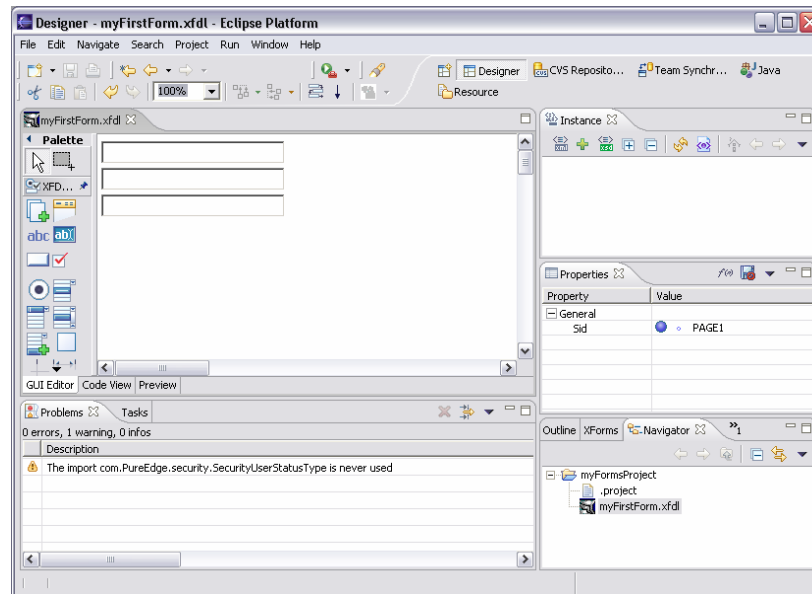
### How Do I Import an Existing File Into an Existing Project?

Sometimes files have already been created and you want to import them into a new project. Import an existing file into a project by right-clicking the project (in the Navigator View) and selecting **Import**.

There are several different locations from which a file can be imported; select the import source and click **Next**. The dialogs will be different for each import source and will not be covered in this document. For more information please refer to the Eclipse Help.

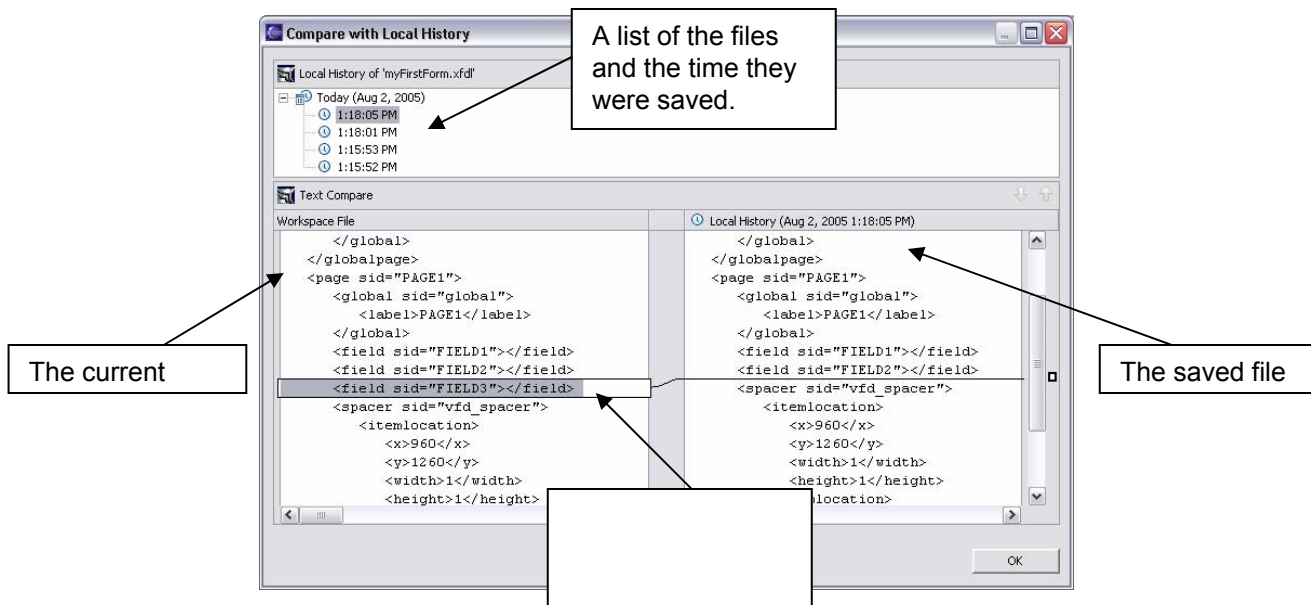






### 9.4.3. Comparing Your Current File Against the Local History?

Once it is open you can access the local history for any file in the current project. Right-click the desired file and choose **Compare With > Local History**.



### 9.4.4. How Do I Replace a File With One From the Local History?

There is a procedure that if after changing a file you determine that you want to revert back to one of the saved editions stored in the local history. To replace a file with one from the local history, right-click the file and select **Replace With > Local History...** select the desired edition and then press the **Replace** button.

## 9.5. Problems View

For information about the Problems View please refer to the Eclipse help, which can be found by selecting **Help > Help Contents** from the **File** menu and entering “problems view” as the search criteria.

## 9.6. Tasks View

For information about the Tasks View please refer to the Eclipse help, which can be found by selecting **Help > Help Contents** from the **File** menu and entering “tasks view” as the search criteria.

## Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation Office  
4360 One Rogers Street  
Cambridge, MA 02142  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## **Trademarks**

IBM, the IBM logo, Workplace Forms, DB2, and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.