



**User's Guide**

**Note**

Before using this information and the product it supports, read the information in "Notices," on page 201.

**First Edition (September 2006)**

This edition applies to version 2.6.1 of IBM Workplace Forms Designer (product number L-DSED-6RFRFB) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces version 2.6 of Workplace Forms Designer.

© Copyright International Business Machines Corporation 2003, 2006. All rights reserved.

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

---

# Contents

<b>Introduction</b> . . . . .	<b>1</b>	Adding pages to a form . . . . .	25
<b>Getting started</b> . . . . .	<b>3</b>	Moving between pages . . . . .	26
Planning a form . . . . .	3	Moving to any page . . . . .	26
Designing a form . . . . .	4	Moving to the next page . . . . .	26
Moving to the previous page . . . . .		Moving to the previous page . . . . .	26
<b>Interface</b> . . . . .	<b>5</b>	Sizing pages . . . . .	26
Perspectives . . . . .	5	Setting page properties . . . . .	26
Selecting a perspective . . . . .	5	Setting properties for a single page . . . . .	27
Editor . . . . .	6	Setting properties for all pages . . . . .	27
Design panel . . . . .	6	Ordering pages . . . . .	27
Source panel . . . . .	6	Deleting pages . . . . .	27
Preview panel . . . . .	7	Providing navigation between pages . . . . .	28
Palette . . . . .	7	<b>Laying out items on a page</b> . . . . .	<b>29</b>
Pinning Palette libraries . . . . .	8	Item types . . . . .	29
Hiding and showing the Palette . . . . .	8	Standard Library items . . . . .	29
Views . . . . .	8	Object Library items . . . . .	35
Showing, minimizing, maximizing or closing		Inserting items onto a page . . . . .	37
views . . . . .	8	Arranging items on a page . . . . .	37
Navigator view . . . . .	9	Bounding boxes . . . . .	37
Outline view . . . . .	9	Layout grids . . . . .	38
Properties view . . . . .	9	Rulers and guides . . . . .	39
Problems view . . . . .	9	Zooming and display . . . . .	40
Enclosures view . . . . .	10	Selecting items . . . . .	41
Advanced views . . . . .	10	Moving items . . . . .	41
Resizing items . . . . .		Resizing items . . . . .	41
Changing the build order of items . . . . .		Changing the build order of items . . . . .	42
<b>Projects</b> . . . . .	<b>13</b>	Tab order . . . . .	42
Creating projects . . . . .	13	Changing the tab order of items on a page . . . . .	42
Opening and closing projects . . . . .	13	Aligning items . . . . .	43
Opening projects . . . . .	13	Alignment types . . . . .	44
Closing projects . . . . .	13	Spacing items . . . . .	45
Creating project sub-folders . . . . .	14	Removing relative alignment assignment . . . . .	46
Linking resources to projects . . . . .	14	Expanding items . . . . .	46
Renaming projects . . . . .	14	Expansion types . . . . .	46
Deleting projects . . . . .	15	Cutting, copying, pasting and deleting items . . . . .	47
Cutting items . . . . .		Cutting items . . . . .	48
Copying items . . . . .		Copying items . . . . .	48
Pasting items . . . . .		Pasting items . . . . .	48
Deleting items . . . . .		Deleting items . . . . .	48
Visibility . . . . .		Visibility . . . . .	48
Converting XFDL items to XForms . . . . .		Converting XFDL items to XForms . . . . .	49
<b>Forms</b> . . . . .	<b>17</b>	<b>Setting item properties</b> . . . . .	<b>51</b>
Creating forms . . . . .	17	Resetting a property to its default value . . . . .	51
Opening forms . . . . .	17	Copying a property setting from one item to	
Upgrading forms . . . . .	18	another . . . . .	51
Saving forms . . . . .	18	Changing colors . . . . .	52
Saving a form with a new name . . . . .	19	Changing the background and text color . . . . .	52
Compressing forms . . . . .	19	Changing the toolbar color . . . . .	52
Renaming forms . . . . .	19	Changing the color of an item . . . . .	53
Setting global form properties . . . . .	19	Showing or hiding categories . . . . .	53
Creating form templates . . . . .	20	Showing advanced properties . . . . .	53
Customize your template images and descriptions		Sorting properties in alphabetical order . . . . .	54
Recovering previous versions of a form . . . . .	21	Help messages . . . . .	54
Closing forms . . . . .	21		
Deleting forms . . . . .	22		
Importing forms . . . . .	22		
Exporting forms . . . . .	22		
<b>Pages</b> . . . . .	<b>25</b>		

Adding context-sensitive help to an item . . . . .	54
Creating an error message for a field or combo box . . . . .	55
Adding accessibility messages . . . . .	55

**Toolbars . . . . . 57**

Adding a toolbar to a page . . . . .	57
Adding items to a toolbar . . . . .	57
Resizing a toolbar . . . . .	57
Copying a toolbar from one page to another . . . . .	58

**Fields . . . . . 59**

Creating a field . . . . .	59
Creating a field with a label . . . . .	59
Specifying the type of data to accept . . . . .	59
Data types. . . . .	60
Specifying the constraints on data . . . . .	61
Constraints types . . . . .	61
Setting up a mandatory field . . . . .	62
Reformatting input data . . . . .	63
Presentation types . . . . .	63
Changing scroll bars . . . . .	66

**Buttons and actions . . . . . 67**

Buttons. . . . .	67
Creating buttons. . . . .	68
Creating submit buttons . . . . .	68
Creating link or replace buttons . . . . .	71
Creating save or cancel buttons. . . . .	71
Creating print buttons. . . . .	71
Automatic actions . . . . .	71
Creating automatic actions . . . . .	72

**Lists and choices . . . . . 75**

Creating check box lists . . . . .	76
Creating radio lists . . . . .	77
Creating popup lists . . . . .	77
Creating combo box lists . . . . .	78
Creating box lists . . . . .	79
Creating calendars . . . . .	80
Using lists to trigger actions. . . . .	80

**Images. . . . . 81**

Adding an image file to a form. . . . .	81
Adding an image to a button or label. . . . .	82
Resizing and cropping images on buttons and labels . . . . .	82
Image-mapping a button . . . . .	83
Adding a background template image . . . . .	83

**Formulas. . . . . 85**

When to use formulas . . . . .	85
Planning a formula . . . . .	85
Setting up simple formulas . . . . .	86
Setting one value to equal another (assignment)	86
Performing a calculation based on two values (calculation) . . . . .	86
Summing values. . . . .	88
Setting an item value to equal a function . . . . .	88

Making decisions based on user input (if/then/else). . . . .	89
--	----

Functions . . . . .	91
Operators and the order of operations . . . . .	96
Creating custom formulas . . . . .	97

References: Referring to other items and their options . . . . .	97
Deleting a formula . . . . .	98
Formula examples . . . . .	98
Automatically calculating compound interest factor . . . . .	98
Displaying the current date automatically . . . . .	100

**Custom functions . . . . . 101**

Creating custom functions . . . . .	101
Making custom functions available . . . . .	101
Distributing IFX files . . . . .	102
Embedding JAR file . . . . .	102
Using custom functions in the Compute Wizard . . . . .	102

**Attachments. . . . . 105**

Attaching files to a form. . . . .	105
Creating attachment buttons . . . . .	106

**Signatures . . . . . 107**

Signing a form . . . . .	108
Signature types. . . . .	109
Digital signatures . . . . .	109
Generic RSA signatures . . . . .	109
Entrust signatures . . . . .	110
Microsoft CryptoAPI signatures . . . . .	110
Netscape signatures . . . . .	110
Signature Pad signatures . . . . .	110
Silanis signatures . . . . .	111
Clickwrap signatures . . . . .	111
Authenticated Clickwrap signatures . . . . .	112
Creating signature buttons . . . . .	112
Creating a Generic RSA signature button . . . . .	112
Creating a Microsoft CryptoAPI signature button . . . . .	113
Creating a Clickwrap signature button . . . . .	114
Creating an Authenticated Clickwrap signature button . . . . .	115
Creating an Entrust signature button . . . . .	116
Creating a Netscape signature button . . . . .	116
Creating a Signature Pad signature button. . . . .	117
Creating a Silanis signature button . . . . .	119
Signing portions of forms . . . . .	121
Specifying the display for a signature button . . . . .	124
Making a signature button mandatory . . . . .	125
Signature properties . . . . .	125

**Web services . . . . . 127**

Adding Web services to a form . . . . .	127
Deleting an enclosed WSDL file from a form . . . . .	127

**XForms . . . . . 129**

Adding XForms support. . . . .	130
Adding XForms support to a new form . . . . .	130
Adding XForms support to an existing form . . . . .	130

The XForms model . . . . .	130
Naming an XForms model . . . . .	131
XForms data instances . . . . .	131
Creating an XForms data instance . . . . .	132
Building XForms data instances . . . . .	134
XForms user interface . . . . .	138
XForms items . . . . .	138
Creating XForms labels . . . . .	139
XForms fields . . . . .	140
XForms lists . . . . .	141
XForms conditional items . . . . .	148
XForms tables . . . . .	151
XForms help messages . . . . .	159
Converting XFDL items to XForms items . . . . .	160
XForms binding . . . . .	161
Binding using ref or nodeset . . . . .	161
Binding using bind . . . . .	162
XForms model binds . . . . .	162
Highlighting bound XForms items . . . . .	164
XForms submissions . . . . .	165
Adding submissions to an XForms model . . . . .	165
Adding a submission to an XForms data instance . . . . .	165
Naming submissions . . . . .	165
Setting which data is submitted . . . . .	165
XForms Smartfill . . . . .	169
<b>XML Model . . . . .</b>	<b>173</b>
Displaying XML Model views . . . . .	174
Creating an XML Model . . . . .	174
Adding an XML Model to a form . . . . .	174
Adding data instances to the XML Model . . . . .	174
Naming data instances . . . . .	175
Deleting a data instance . . . . .	175
Designing data instances . . . . .	175
Adding elements to a data instance . . . . .	176
Renaming data instance elements . . . . .	176
Adding child elements to an element . . . . .	176
Deleting data instance elements . . . . .	176

Changing the namespace of data instance elements . . . . .	177
Adding attributes to data instance elements . . . . .	177
Renaming attributes . . . . .	177
Adding a value to an attribute . . . . .	177
Converting an attribute to a namespace attribute . . . . .	177
XML Model binding . . . . .	177
Binding a data instance node to the form . . . . .	178
Deleting the bindings for an instance . . . . .	178
XML Model submissions . . . . .	178
Adding submissions to an XML Model . . . . .	178
Setting the submission rules . . . . .	178
Deleting submissions . . . . .	179
Submission properties . . . . .	179
Adding an XML submission button . . . . .	179

<b>Customizing the Designer interface . . . . .</b>	<b>181</b>
Customizing hot keys . . . . .	181
Customizing the Palette . . . . .	181
Selecting a Palette layout . . . . .	181
Using large button icons . . . . .	181
Creating your own custom library . . . . .	182
Creating user defined perspectives . . . . .	182
Exporting objects . . . . .	183

<b>Appendix A: Accessibility . . . . .</b>	<b>185</b>
Keyboard input and navigation . . . . .	185
Keyboard shortcut keys . . . . .	185
Features for accessibility display . . . . .	185
Accessible documentation . . . . .	185

<b>Appendix B: Options . . . . .</b>	<b>187</b>
--------------------------------------	------------

<b>Appendix. Notices . . . . .</b>	<b>201</b>
Trademarks . . . . .	202

<b>Index . . . . .</b>	<b>203</b>
------------------------	------------



---

## Introduction

IBM® Workplace Forms™ Designer allows form designers to create XFDL forms within both a graphical drag-and-drop environment and a powerful source code editor. You use the Designer together with IBM Workplace Forms Viewer. The Viewer lets users view and complete XFDL forms. When designing a form, you use the Viewer to preview and test your form.

The Designer is based on the Eclipse platform. The Eclipse platform is an environment for developing and delivering software applications. The overall Eclipse interface is referred to as the *Workbench*. If you have previously used the Eclipse Workbench, then the Designer will look very familiar and you will already know how to perform many general functions. If you have not previously used the Eclipse Workbench, be aware that certain general functions in the Designer are common to all Eclipse-based software applications. These general functions are not described in detail in this document. For detailed information about the Eclipse Workbench, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).





---

## Getting started

This section describes how to get started using the Designer for the first time. Before you can start working on a form, you must first create:

- A **workspace** — A single directory where the projects, folders and files that you create in the Designer are all stored. To open the Designer, you must first specify the workspace.
- A **project** — Contains folders and files, and they can be opened, closed, or built. When you open the Designer, the project's contents are displayed in the Navigator view.
- A **form file** — An XML document that conforms to the XFDL specification.

To start using the Designer:

1. Click **Start** → **All Programs** → **IBM Workplace Forms Designer 2.6** → **IBM Workplace Forms Designer**.

The Workspace Launcher window opens. You use this window to set up your default workspace.

2. Click **OK** to accept the default workspace. You can change your workspace at a later time.

**Note:** The Workspace Launcher window will appear every time you start the Designer. If you do not want this window to appear, click **Use this as the default and do not ask again**. The next time you start the Designer, this window will not be displayed.

The Designer's Welcome view opens.

**Note:** The Designer Welcome page uses your browser to display content. You must have Internet Explorer 5.5 or higher or a Mozilla-based browser for the Welcome page to display.

3. In the Welcome view, click the **Create Form** button to open the New Workplace Form window.
4. Enter a **File Name** for the new form.
5. Click **Finish** to create the form.  
A message appears asking if you want to switch to the Designer perspective.
6. Click **Yes**.

The Designer opens and the form is displayed in the **Design** editor.

Now, you are ready to build the form.

---

## Planning a form

Before you begin designing a form, there are many things you need to consider:

- What information will the form collect?
- Are you duplicating an existing form (for example, a paper form)?
- In what order should the user enter the information?
- Will items be positioned on the page using absolute positioning or relative positioning?
- Does the form need distinct sections? If so, how many?

- Should the form have more than one page? If so, what should go on each page?
- What page size do you require?
- Will the user need to print the form?
- Will the form need to verify user input (for example, for correct data type or format) as the user fills out the form?
- Will the form need to perform calculations as the user fills out the form?
- What security features are necessary?
- Should the form support digital signatures?
- To whom or to where will users submit the form? Will any user input be submitted into a database?
- When the form opens, will items need to be updated or pre-populated (for example, by a database or via Smartfill)?
- Will you deliver the form to users via IBM Workplace Forms Server - WebForm Server?
- Will you use a wizard style form?
- Should the form contain dynamic elements that appear or disappear as needed?
- How should the form connect to the rest of your application?

---

## Designing a form

You can design a form in various ways:

- Create an entirely new form in the Designer (see below).
- Start with an existing paper form, scan it, and recreate it in the Designer using a template image (see “Adding a background template image” on page 83).
- Start with a template form and modify/customize it in the Designer (see “Creating forms” on page 17).
- Start with an existing sample Workplace Form and modify/customize it in the Designer (see “Opening forms” on page 17 and “Upgrading forms” on page 18).

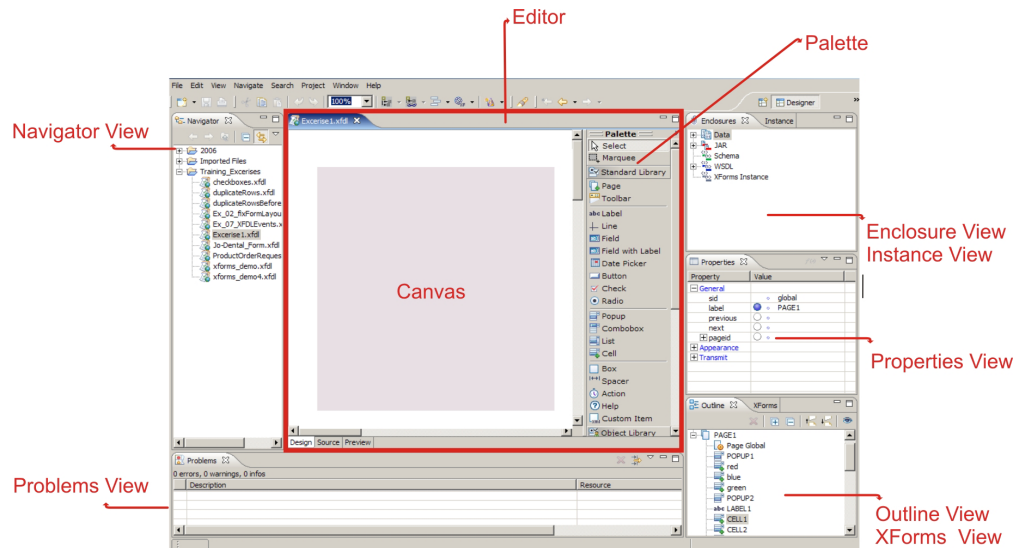
To create an entirely new form:

1. Create a project for your form. See “Projects” on page 13.
2. Create a blank form. See “Forms” on page 17.
3. Layout your form by inserting items onto the form. See “Laying out items on a page” on page 29.
4. Set item properties to control their appearance and behavior. See “Setting item properties” on page 51.
5. Preview the form in the Preview panel or in the Viewer. See “Editor” on page 6.

---

## Interface

This section describes the parts of the Designer interface.



---

## Perspectives

The Designer interface is comprised of several *views* such as the Instance and Navigator views and an editor pane where you build forms. Taken together, these views and the editor pane make up the Designer perspective.

A *perspective* is a combination of Eclipse Workbench views and editors that lets you view and work with files. The Eclipse Workbench comes with several perspectives and, much like the word implies, each perspective lets you view and manage your work from a different angle, providing specific functionality intended to accomplish particular tasks.

The Designer comes with two perspectives that are available by default: Designer and Resource. The Designer perspective displays the views that are required to build a form. The Resource perspective displays the views that are required to manage your projects and files.

Perspectives control the contents of certain menus and toolbars. They define visible action sets, that you can change to customize a perspective. You can save a customized perspective. For detailed information about perspectives, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

**Tip:** If you change a perspective by mistake, you can revert to the Designer perspective by clicking **Window** → **Open Perspective** → **Designer**.

### Selecting a perspective

Within the Designer, you can switch perspectives.

**Note:** Since the **Designer** perspective contains all the views and editors you need to create and design forms, until you become comfortable working with the Designer, you should work exclusively from the **Designer** perspective.

To select a perspective:

1. Click **Window** → **Open Perspective** → **Other** to open the Select Perspective window.
2. Select a perspective from the list.
3. Click **OK**.

Designer displays your project from the perspective you selected.

To revert back to the **Designer** perspective, click **Window** → **Open Perspective** → **Designer**.

---

## Editor

When you create or open a form, it is displayed in the editor. The editor is comprised of the following three tabbed panels:

- **Design** — A visual editor that you use primarily to design the appearance of your form; this editor includes the canvas and the Palette.
- **Source** — A text editor that lets you view and edit your form's source code.
- **Preview** — A form preview that lets you preview and test your form in an embedded Viewer.

**Note:** You must have the Viewer installed to use the **Preview** panel.

You use these panels to create, edit and view forms. The panels can be displayed by selecting the tab at the bottom of the editor.

### Design panel

Use the Design panel to create and edit the visual components of your forms. The Design panel consists of two parts:

- the Canvas — The area where you design the visual components of your forms.
- the Palette — Contains the items you can add to the form.

### Source panel

Use the Source panel to view and edit your form's XFDL source code.

**Note:** The Source panel has code validation; if your code has errors, in some cases you will not be able to go back to the Design panel until the errors are resolved.

When you switch from the Source panel to the Design panel, the Designer validates the form against the XFDL schema. If the form does not comply with the schema, the Designer prevents you from leaving the Source panel and displays a list of errors in the Problems view.

For example, each option type is valid for some item types and invalid for other item types. If you use the Source panel to change an item's type and do not ensure that all of its options are still valid for the item type, the form will no longer comply with the XFDL schema and you will not be able to switch back to the Design panel.

For detailed information on the Problems view, see “Problems view” on page 9.

## Editing source code using code assist

*Code assist* provides you with a list of code options that are available to you at that particular point in your code. You can use code assist when writing or editing XFDL source code.

To use code assist:

1. Place your cursor in a valid position on a line of code and then press Ctrl + space bar. **Note:** On Chinese operating systems, use Alt+./.

If the Designer finds valid code completion for this position, code assist displays a list of possible suggestions in a popup window.

2. If there are multiple suggestions, you can type additional letters to narrow the list.
3. Double-click the desired code fragment.

The code fragment is added to your form’s source code.

## Preview panel

Use the Preview panel to view and test your forms. When you click the **Preview** tab, the form opens in the Viewer.

**Note:**

- If you use a non-OpenType font, a non-TrueType font, or a vertical font in your form, the Designer may display a warning message saying the font is not available. You can ignore this message.
- Do not use the **Save Form** or **Save As** toolbar buttons in the Preview to save the form within your workspace.

---

## Palette

The Palette contains tools that let you create and select items.

### Selection tools

For detailed information about the selection tools, see “Selecting items” on page 41.

### Libraries

Creation tools are grouped within libraries. The Palette contains two default libraries:

- **Standard Library** — Contains tools that let you create simple form items.
- **Object Library** — Contains tools that let you create complex pre-defined XFDL objects that are composed of several items. For more details, see “Object Library items” on page 35.

You can collapse or expand a library by clicking on the library name. For detailed information about customizing the Palette, see “Customizing the Palette” on page 181.

## Pinning Palette libraries

Palette libraries expand or collapse dynamically. If you open a library, other libraries automatically close. If you do not want a library to close, you can pin a library so that it does not collapse when you expand another library.

To pin a library:

Click the library.

The library will now remain expanded.

## Hiding and showing the Palette

You can hide the Palette, giving you more space in the editor.

To hide the Palette:

On the Palette title bar, click .

The Palette minimizes.

To show the Palette:

On the Palette title bar, click .


---

## Views

The Eclipse Workbench interface is comprised of views and an editor pane.

A *view* is a tabbed window that groups similar information together. For example, the Properties view is a context-sensitive view that displays the properties of whatever object is selected in the Designer, the Navigator view lists all the files and folders associated with your project, and the Enclosures view displays all the files that are enclosed within the form.

A view might appear by itself, or tabbed with other views.



Views have their own menus. To open the menu for a view, click  located to the right of that view's title bar. Some views have their own toolbars.

You can open and close views, and dock them in different positions in the Workbench.

This section describes the main Designer views. For detailed information about other views, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

## Showing, minimizing, maximizing or closing views

You can show, minimize, maximize, or close views.

- To **show a view**, click **Window** → **Show view**.
- To **minimize or maximize a view window**, click  or  located at the right of the view toolbar.

- To **maximize the view**, in the Workbench window double-click a view title bar.
- To **close a view**, click the X in the view tab.

## Navigator view

The Navigator view lists the folders and files in your project.

Using this view, you can open files, copy, move or create new resources, select resources for importing or exporting, and compare and replace resources. Most of these operations can be accessed by right-clicking in the Navigator view.

For detailed information about the Navigator, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

## Outline view

The Outline view displays the hierarchical structure of your form. You can expand or contract the outline to see more or less detail in the form.

Clicking an item in the Outline view highlights the item on the canvas or in the Source panel, and displays its options in the Properties view.

**Tip:** The Outline view is very useful for accessing non-visible items in a form, such as form and page globals. You can also use the Outline view to move to pages in a multi-page form.

The Outline view's contents and toolbar will vary depending on whether you are working in the Design panel or Source panel.

For detailed information about the Outline view, see *Outline view* in the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

## Properties view

The Properties view displays the properties you can set for a selected form item or object. The list of properties will vary depending on what is selected in the Designer.

For detailed information about setting item properties, see "Setting item properties" on page 51.

For detailed information about specific properties, see "Appendix B: Options" on page 187.

## Problems view

The Properties view displays errors, warnings, and other information whenever you check your form or switch from the Source panel to the Design panel.

**errors** Errors are problems that make the form impossible to use. The Viewer cannot open forms with these errors. When you receive an error message, you will also be given information about which of the form's items or settings is causing the problem.

### warnings

Warnings are reported when the Designer finds an item or property setting that is either incorrect or not part of standard XFDL. For example, *The following problem was detected* or *Referenced item does not exist*. The form can be opened with these types of problems.

**infos** General tips and tricks information.

For detailed information about the Problems view, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

## Enclosures view

You can use the Enclosures view to enclose files within your form:

**Data** The Designer lets you specify which page of your form you can attach a file to. The Data option lets you select the appropriate page. Images or documents to be used with attachment features are added here. For detailed information about adding images, see “Adding an image file to a form” on page 81. For detailed information about attachments, see “Attachments” on page 105.

**JAR** The Designer lets you enclose custom Java™ modules containing additional form functions. For detailed information on jar files, see “Custom functions” on page 101. For information about how to develop your own Java modules (.jar files), see the *Java API User’s Manual*.

### Schema

The Designer lets you enclose a schema file in your form. You can then use the enclosed schema to create instances. You can also use an enclosed schema to validate the information in the model. The schema validates all data in the model in the target namespace for that schema.

### WSDL

The Designer lets you enclose a Web Services Definition Language (WSDL) document to your form. Once enclosed, you can use the WSDL to generate an instance.

For detailed information about Web Services, see “Web services” on page 127.

### XForms Instances

The Designer lets you enclose an XML file that contains an XForms instance.

## Advanced views

The following advanced views are available:

### Instance view

The Instance view is used to define the XML template for the data that will be collected from the form. A data instance can be used to store input values, pre-populated fields with data, or generate list selections.

For detailed information on instances, see “XForms data instances” on page 131.

### XForms view

The XForms view is used to add XForms support and to manage XForms models. Once you create an XForms model, you can use the XForms view to add, edit or delete the XForms elements such as XForms models, instances, submissions, binds, and schemas.

For detailed information about XForms, see “XForms” on page 129.



**XML Model view**

You use the XML Model view to create and manage an XML Model.

**Note:** The XML Model view is not part of the Designer perspective.

For detailed information about the XML model, see “XML Model” on page 173.

**XML Model Instance view**

The XML Model Instance view provides a listing of the XML data instances used in your form.

**Note:** The XML Model Instance view is not part of the Designer perspective.

For more information about XML Model data instances, see “Adding data instances to the XML Model” on page 174.



---

## Projects

Projects are a way of organizing forms and resources.

Like folders, projects map to directories in the file system. (When you create a project, you specify a location for it in the file system.)

For detailed information about projects, see *Working with projects, files and folders* in the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

---

### Creating projects

To create a project:

1. Click **File** → **New** → **Project**.
2. Expand **Simple** and click **Project**.
3. Click **Next**.
4. Click within the **Project name** field.
5. Type the name of the project.
6. Specify where the **Project contents** will be saved:
  - To save your project into your default Workspace, select the **Use default** checkbox.
  - To select any directory, clear the **Use default** check box and click **Browse**.
7. Click **Finish**.

---

### Opening and closing projects

A project is either open or closed. When a project is closed, it cannot be changed. The resources of a closed project will not appear in the Navigator view, but the resources continue to reside on the local file system. Closed projects require less memory. When a project is open, the structure of the project can be changed and you will see the contents.

In the Navigator view, the project's icon is either an open folder or a closed folder.

#### Opening projects

To open a closed project:

1. In the Navigator view, select the project.
2. Select **Projects** → **Open Project**.

The project's icon changes to an open folder.

Alternatively, you can right-click the project and select **Open Project**.

#### Closing projects

To close a project:

1. In the Navigator view, select a project folder.
2. Click **Projects** → **Close Project**.

The project's icon changes to a closed folder.

Alternatively, you can right-click the project listed and select **Close Project**.

---

## Creating project sub-folders

To create a folder within a project:

1. Click **File** → **New Folder**.
2. Select the parent folder from the list provided.
3. Click within the **Folder name** field.
4. Type the folder name.
5. Click **Advanced**.
6. Click **Link to folder in the file system**.
7. Type a file system path, or click **Browse** to select a folder in the file system.
8. Click **Finish**.

---

## Linking resources to projects

Folders and files directly below projects can be linked to locations in the file system outside of the project's location. These special folders and files are called linked resources.

To create a linked folder:

1. In the Navigator view, right-click the project in which you want to link to other resources and click **New** → **Other**. A **Select a wizard** window is displayed.
2. Expand **Simple** and click **Folder**.
3. Click **Next**.
4. Click within the **Folder name** field.
5. Type the name of the folder as it will appear in the workbench. This name can be different from the name of the folder in the file system.
6. Click **Advanced**.
7. Click **Link to folder in the file system**.
8. Type a file system path, or click **Browse** to select a folder in the file system.
9. Click **Finish**.

To create a linked file, follow the same steps as above, except choose **New** → **File**.

For detailed information about linked resources, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

---

## Renaming projects

To rename a project:

1. In the Navigator view, select the project.
2. Click **File** → **Rename**.  
The project name is highlighted.
3. Type the new project name and press **Enter**.

Alternatively, right-click the project and select **Rename Project**.

**Note:** Projects are listed alphabetically in the Navigator view.

---

## Deleting projects

To delete a project:

1. In the Navigator view, select a project to delete.
2. Click **Edit** → **Delete**. Alternatively, right-click the project and click **Delete Project**.
3. Select one of the following:
  - **Also delete contents under “project path”.**
  - **Do not delete contents.**
4. Click **Yes**.



---

## Forms

A Workplace Form is an electronic document or interface for collecting information from people.

**Note:** Do not edit forms externally (for example, using a text editor) if the form is part of a Designer project or is saved within a Designer workspace. If you want to edit a form using a text editor, save the form to another location, and remove it from any Designer projects or workspaces.

---

## Creating forms

To create a form:

1. Select **File** → **New** → **New Workplace Form** to open the New Workplace Form window.

**Note:** **File** → **New** → **New Workplace Form** is only available in the Designer perspective. To create a form in any perspective, select **File** → **New** → **Other** and select the **Workplace Forms** → **New Workplace Form** wizard.

2. Select the project for the new form.
3. In the **File name** field, type the name for the new form.
4. Click **Next**. The **Choose Template** window opens.
5. Within the **Choose Template** field, select a form template.
6. Click **Finish**.

The form with a watermark titled “Empty” is displayed in the canvas and listed in the Navigator view. (You may need to expand the project in the Navigator view to see the list of forms in the project.)

For detailed information on creating form templates, see “Creating form templates” on page 20.

For detailed information about XForms, see “XForms” on page 129.

---

## Opening forms

Opening a form displays the form in an Editor, allowing you to edit and preview the form.

To open a form:

In the Navigator view, double-click the form name. If the form is not listed in the Navigator view, you must import the form into a project (see “Importing forms” on page 22).

If you open a form that is based on XFDL 6.x, the Designer will ask if you want to upgrade the form to XFDL 7.0. You have the option of making a copy of the older form and saving it to another location.

---

## Upgrading forms

You can use the Designer to create and edit forms. Forms created in the Designer are based on XFDL 7.0. To edit a form that is based on an earlier version of XFDL (for example, a form created in an older version of the Designer), you must upgrade the form to XFDL 7.0. You can only upgrade forms based on XFDL 6.0 or higher.

**Note:** The Designer may encounter difficulty upgrading forms with complicated formulas or custom information. Make sure you thoroughly test all upgraded forms.

To upgrade a form:

1. Select **File** → **New** → **Upgrade Workplace Form** to open the Upgrade Workplace Form window.

**Note:** **File** → **New** → **Upgrade Workplace Form** is only available in the Designer perspective. To upgrade a form in any perspective, select **File** → **New** → **Other** and select the **Workplace Forms** → **Upgrade Workplace Form** wizard.

2. Select the forms to upgrade:
  - To select form files, select **Files in the File System** and click **Browse** to select the forms.
  - To select a directory of forms, select **A Directory in the File System** and click **Browse** to select the directory.
  - To select forms from your projects, select **Your Workspace** and use the navigation tree to select the forms.
3. Click **Next** to display the next page of the wizard.
4. Select the project for the new form.
5. In the **File name** field, type the name for the new form. The default filename is the original filename plus `_V70.xfdl`.
6. If you want to open the new form, select the **Open Form(s) After Finishing** checkbox.
7. If you want to overwrite an existing form, select the **Overwrite existing form** checkbox.
8. Click **Finish**.

The Designer lists the upgraded form in the Navigator view. (You may need to expand the project in the Navigator view to see the list of forms in the project.)

---

## Saving forms

You can only save a form that has been edited since its last save. The Designer indicates that a form has been modified by displaying an asterisk "\*" beside the form name in the Editor tab.

To save a form:

With the form open and active in the Editor, click **File** → **Save**.



---

## Saving a form with a new name

To save a form with a new name:

1. With the form open and active in the Editor, click **File** → **Save As** to open the Save As window.
2. Select a project for the form.
3. In the **File name** field, type a new name for the form.
4. Click **Finish**.


The Designer lists the new form in the Navigator view, closes the original form in the Editor and opens the new form in the Editor. (You may need to expand the project in the Navigator view to see the list of forms in the project.)

---

## Compressing forms

By default, when you save a form it is saved as an uncompressed XFDL file. You can also setup your form so it is saved as a compressed XFDL file. Compressed XFDL files are compressed using a modified gzip format; this format is unique to Workplace Forms. You cannot view a compressed file in a text editor, but you can use the form in the Viewer and the Designer. Other compression software may not be able to decompress the forms. You can use an uncompressed file in the Viewer, the Designer, or a text editor.

To compress a form:

1. In the Outline view, expand **globalpage** and select **Form Global**.
2. In the top-right corner of the Properties view, click  and click **Show Advanced Properties**.
3. Expand **Miscellaneous**.
4. Click within the **saveformat** value field.
5. Type `application/vnd.xfdl;content-encoding="base64-gzip"` and press Enter.

The next time you save the form, it will be saved as a compressed XFDL file.

---

## Renaming forms

To rename a form:

1. In the Navigator view, select the form name.
2. Click **File** → **Rename**.
3. Type a new name for the form and press Enter.

---

## Setting global form properties

You can set the default properties for all pages and all items in a form by setting the global form properties.

For example, if you want all items on your form to use the same font, you could set the font in the global form properties. Whenever you insert an item onto your form, it will have the default font specified in the global form properties.

The global form properties only affect the *default* properties of items. You can later modify the properties of individual items.

To set global form properties:

1. In the Outline view, expand **globalpage** (at the top of the list) and select **Form Global**.
2. In the Properties view, set the form's global page properties. For detailed information on setting properties, see "Setting item properties" on page 51.

---

## Creating form templates

You can use forms as custom templates and share them with team members to create several forms using the same layout and functionality.

To setup custom form templates:

1. Click **Window** → **Preferences** to open the Preferences window.
2. In the left column, expand **Workplace Forms**.
3. Click **Form Templates**.
4. Click **New** to open the Browse For Folder window.
5. Browse to the directory of forms you want to use as templates and click **OK**.  
You can use template forms saved to your own computer or stored on a shared network drive.
6. Click **Apply** and then **OK**. The forms in the directories you have selected will display in the **New Workplace Forms** wizard.
7. If you want to change the order that the form templates are listed in the **New Workplace Forms** wizard, select the folder and click **Up** and **Down**.
8. If you want to remove a directory of templates, select it and click **Remove**.
9. If you want to use only the default templates, click **Restore Defaults**.
10. Click **OK** to close the Preference window.

---

## Customize your template images and descriptions

When you add a new template to the Designer, you can also add a description and a preview image of the form. These steps are optional, but can help you to quickly identify each template if you are distributing your templates to other users in your organization.

To add a new preview image:

1. Take a snapshot of your form.  
You can do this using a variety of screen-capturing tools.
2. Size the image to 160 x 240 pixels.  
The Designer will automatically scale any image to this size, but you will get better image quality if you resize the image yourself before adding it to the Designer.
3. Save the image so that the filename is the same as your template, and use one of the following formats: jpg/jpeg, gif, bmp, png, wbmp.  
For example, if your template is named `governmentTemplate.xfd`, save the image as `governmentTemplate.jpg`.
4. Copy the image to the same template folder that contains the form it represents.

Once you have added the image file, the Designer will automatically display this image when the user selects the corresponding template.

To add a description to your own template.

To add a description to your template, you need to create a separate file that will contain this information. This file has the following name:

`form_file_name.properties`

For example, `governmentTemplate.properties`

The file includes two settings:

- `TemplateTitle` — The title of the template form. You should limit this to a few words.
- `TemplateDescription` — A text description of the form. This can be several lines long.

The properties file is written as a standard text file using tag-value pairs, as shown:

```
TemplateTitle=your template name
TemplateDescription=Use this template for all government forms.
                    It includes a toolbar and page navigation buttons.
```

Once you have created the properties file, copy it to the same template folder that contains the form it describes. The Designer will access this file automatically, and will display the description of the form when the user selects that template.

---

## Recovering previous versions of a form

You can recover previous versions of a form by using local history.

*Local history* is a local file source control mechanism. Each time you edit and save a file in the Designer, a copy of it is saved. This lets you compare your current form to an older version, or replace the file with a previous version. Each version in the local history is identified by the date and time the file was saved. This is helpful if you delete a portion of your form that you want to restore, or to identify how your form has changed.

For detailed information about local history, see *Comparing resources with local history* in the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

To recover a previous version of a form:

1. Within the Design or Source editor, right-click and click **Replace With** → **Local History**.
2. Select the version and click **Replace**.

**Note:** You can also access local history for your file from the Navigator view. Right-click the desired file and click **Compare With** → **Local History**.

---

## Closing forms

To close a form:

Click the X in the form's Editor tab.

The Designer closes the form in the Editor.

**Note:** To close all open forms, click **File** → **Close All**.

---

## Deleting forms

Before you delete a form, ensure the form is not open in the Editor.

To delete a form:

1. In the Navigator view, select the form name.
2. Click **Edit** → **Delete**.

---

## Importing forms

You can import existing XFDL forms or XForms forms into a project. Importing a form makes it available in the Navigator view and associates it with the project.

Importing an XForms form converts the form into an XFDL form with XForms. Importing an XFDL form does not convert the form in any way. For example, if you are importing an XFDL 6.x form, you will still need to upgrade the form (see “Upgrading forms” on page 18).

To import a form:

1. Click **File** → **Import** to open the Import window.
2. Select the type of item to import:
  - To import an XFDL form, click **File system**.
  - To import an XForms form, click **Import XForms as Workplace Form**.
3. Click **Next** to display the next page of the wizard.
4. To the right of the **From directory** field, click **Browse**.
5. Select the directory that contains the forms to import. The forms in the selected directory are displayed in the list.
6. To the right of the **Into folder** field, click **Browse**.
7. Click **Finish**.

The Designer lists the imported forms in the Navigator view.

**Note:** You can also import forms by dragging them from Windows® Explorer and dropping them onto the project directory in the Navigator view.

---

## Exporting forms

You can export forms from a project. This saves the form as a file to a location outside your project. When you export an XForms form, the Designer strips away all non-XForms elements and creates an XML file containing only XForms elements.

1. Click **File** → **Export**.
2. Select the type of item to export:
  - To export an **XFDL** form, click **File system**.
  - To export an **XForms** form, click **Export Workplace Form as XForms**.
3. Click **Next**.
4. In the left pane, select the form directories you want to export. The list of available forms is updated in the right pane.
5. If you want to specify the files types listed, then click one of the following:
  - **Select By Type** - Reduce selection to only files of specified types.

- **Select All** - Select all files in source directory.
  - **Deselect All**- Deselect all files from source directory.
6. In the right pane, select the forms you want to export.
  7. To the right of **To directory**, click **Browse**.
  8. Select the directory the forms will be exported into and click **OK**.
  9. If you want to specify export details, then set any of the following:
    - Options: **Overwrite existing files without warning**.
    - Options : **Create directory structure for files**.
    - Options: **Create only selected directories**.
  10. Click **Finish**.

The Designer exports the forms.



---

## Pages

When you create a new form, by default it consists of one page. You can add as many pages as you like to a form. However, it is best to keep the form as simple as possible, so try not to add extra pages unless you need to. You may wish to use more than one page for the following reasons:

- **Screen-sized forms** — Typically, users do not like to scroll. It's often a good idea to limit the size of each page to the size of the user's screen. To convert a legal sized paper form into a screen-sized form, you will need to create several pages, and logically group the kinds of information you are collecting onto separate pages.
- **Screen and paper form system** — If you want to provide an easy-to-use on-screen form but also want to print out the collected user input, you can use a multiple-page form to do both. Create the screen-sized forms for collecting information from the user, and then create a final printing page that maps all user input to items on the *printing page*. Then set up the form to print only the *printing page*.
- **Wizard forms** — You may want to create a form that resembles a software application wizard. This type of form consists of a sequence of several small screens, each of which asks the user to answer a few questions.

### What to consider when using more than one page:

When designing a form that uses more than one page, consider the following:

- **Consistency** — Consider providing some consistent elements from page to page to give the user a sense of familiarity. Use the global form properties to set the properties for all pages (for example, fonts, colors, and so on) to maintain consistency. For detailed information on setting page properties, see "Setting page properties" on page 26.
- **Navigation** — To allow the user move from page to page when viewing the form in the Viewer, you must add a paging control (for example, a "Next Page" button) to each page. Consider whether you want to allow both forward and backward paging. Because you control the navigation, you can skip over pages, or even set up the form to decide which page to display next, depending on what the user has entered into the form. For detailed information on providing navigation, see "Providing navigation between pages" on page 28.

---

## Adding pages to a form

To add a page to a form:

1. In the **Palette**, click **Page**.
2. Click anywhere within the canvas or Outline view.

A new page is added to the form. The new page is listed in the Outline view (at the bottom of the list) and is automatically selected and displayed in the canvas. (Only one page can be displayed in the canvas.)

---

## Moving between pages

If your form contains more than one page, you can *move* between pages in the Designer (that is, display a specific page in the canvas).

### Moving to any page

To move to any page:

In the Outline view, select the page.

### Moving to the next page

To move to the next page:

Click **View** → **Next Page**.

### Moving to the previous page

To move to the previous page:

Click **View** → **Previous Page**.

---

## Sizing pages

To size a page:

1. In the Outline view, select the page's **Page Global** item.
2. In the Properties view, expand **Appearance**.
3. Set **pagesize** to the page's width and height (measured in pixels). Approximate values for common page sizes are:
  - A4 (ISO 216) — 932 x 1343
  - B5 (ISO 216) — 800 x 1152
  - Letter (US) — 960 x 1260
  - Legal (US) — 960 x 1620
  - Tabloid (US) — 1260 x 1980
  - PA4 (proposed intermediary between A4 and US letter) — 930 x 1260
  - 640 x 480 (screen display) — 600 x 265
  - 800 x 600 (screen display) — 770 x 375
  - 1024 x 768 (screen display) — 980 x 540

For detailed information about setting properties, see “Setting item properties” on page 51.

---

## Setting page properties

You can modify the appearance and behavior of pages by setting page properties.

Page properties also control the default properties of items that you insert onto pages. For example, if you want all items on a page to use the same font, you could set the font in the page properties. Whenever you insert an item onto the



page, it will have the default font specified in the page properties. The page properties only affect the *default* properties of items. You can later modify the properties of individual items.

Tip: Set the default font for the text on the form in the **Window** → **Preference** settings (**General** > **Appearances** tab). This saves time if the majority of the text on your form will be the same font, size and style

## Setting properties for a single page

When you set the properties for a single page, the settings only apply to that specific page.

To set properties for a single page:

1. In the Outline view, select the page's **Page Global** item.
2. In the Properties view, set the property values. For detailed information about setting properties, see "Setting item properties" on page 51.

## Setting properties for all pages

When you set the properties for all pages, the settings apply to all pages in the form.

**Note:** If you set the properties for all pages *and* set the properties for a specific page, the settings for the specific page will take precedence.

To set properties for all pages:

1. In the Outline view, expand **globalpage**.
2. Click **Form Global**.
3. In the Properties view, set the property values. For detailed information about setting properties, see "Setting item properties" on page 51.

---

## Ordering pages

If your form contains more than one page, you can arrange the order of pages.

To order pages:

1. In the Outline view, select the page and drag it up or down in the hierarchy. When the cursor is at a valid location, a horizontal black line appears in the hierarchy. You can only move a page to a valid location.
2. When the cursor is at the location in the hierarchy where you want the page, release the left mouse button.

---

## Deleting pages

To delete a page:

1. In the Outline view, select the page.
2. In the Outline view toolbar, click the **Delete** button.

---

## Providing navigation between pages

To allow the user to move from page to page when viewing your form in the Viewer, you need to add paging controls. Usually, a paging control will be a button. You need to make the following decisions about paging in your form:

- **Do you want the user to be able to navigate both forward and backward?** Consider whether the user will try to page through the form initially to look at its scope, try to clarify a confusing instruction, or need to fix an error. For example, if the user enters something on page three that invalidates something on page one, the user will need to return to page one to fix the error. Otherwise, the Viewer will not allow the form to be submitted.
- **Do you want to hide pages from the user?** You may want to reformat the user's input for storage or printing on hidden pages in the form.
- **Do you want to show different pages to different users, depending on what they enter into the form?** You can program the form to decide where the user should go next, depending on what has been entered into the form. If you do this, make sure to consider backward navigation as well.
- **Do you want to direct the input focus to an item that is not the first item on a page?** You can program a paging control to direct the input focus to particular items on new pages.

To insert a paging button:

1. In the **Palette**, click **Button**.
2. Click on the canvas to place the button on the page.
3. In the Properties view, expand **General**.
4. Set **type** to **pagedone**.
5. Set **url** to `#PAGE0.global` where `PAGE0` is the name of the page to display (for example, `#PAGE3.global`).

**Note:** To direct the input focus to an item that is not the first item on the page, set **url** to `#PAGE0.ITEM` where `PAGE0` is the name of the page to display and `ITEM` is the name of the item to direct focus to (for example, `#PAGE3.FIELD2`).

You could also provide paging controls using a popup, combobox, or list:

1. Create a popup, combobox, or list.
2. Insert a cell into the popup, combobox, or list.
3. Set the cell's **type** and **url** as described above.

---

## Laying out items on a page

An item is a single element in your form, like a button or field. To make a form, you place visible items on the page. You can also insert hidden items in the form. Hidden items add advanced features to your form.

---

### Item types

A form can include the following general types of items.

**Pages** Each form can be composed of any number of pages, just like a paper form. For detailed information about pages, see “Pages” on page 25.

**Toolbars**

A toolbar is a special area at the top of a form where you can place headings and control buttons. For detailed information about toolbars, see “Toolbars” on page 57.

**Tables and panes (groups)**

Tables and groups provide a way of grouping or associating related items. This makes the data easier to interpret and forms easier to complete. For detailed information about tables, see “XForms tables” on page 151.

**Fields** A field is an area on the form where the user can type in information, such as names, dates, dollar amounts, and so on. For detailed information about fields, see “Fields” on page 59.

**Buttons and actions**

Buttons let the user trigger actions (for example, saving a form). A form can also contain automatic actions that occur without the user explicitly triggering them (for example, submitting data to a database or server every five minutes). For detailed information about buttons, see “Buttons” on page 67.

**Lists and choices**

Lists are a way of presenting choices to users. For detailed information about lists, see “Lists and choices” on page 75.

**Graphics**

Graphics like images, labels, lines, boxes and spacers help you define the visual appearance of a form. For detailed information about images, see “Images” on page 81. For detailed information about spacing items, see “Spacing items” on page 45.

**Objects**

An object is a predefined item or a collection of items (for example, an address block). For more details, see “Object Library items” on page 35 and “Exporting objects” on page 183.

### Standard Library items

If your form contains XForms, then the Standard Library includes tools for creating all XFDL and XForms items. If your form does not contain XForms, then the Standard Library only includes tools for creating XFDL items.

**Action**

Creates an XFDL action that specifies form-initiated actions that executes automatically. For example, you set an action such as *select*, *display*, *print*, *saveform* or *saveas*.

For detailed information about actions, see “Buttons and actions” on page 67.

**Action (Non XForms)**

See **Action**.

**Action (Submit)**

Creates an action associated with an *xforms:submit* option.

**Action (Trigger)**

Creates a button associated with an *xforms:trigger* option.

**Box** Creates a rectangular box on your form to display in the form background. Typically you would set the background color for a box and place it behind a group of items to logically divide items on your page.

**Button**

Creates a trigger button with actions. You can configure which action each button triggers by setting its *type* option and other supporting options. Use buttons to trigger actions such as transmitting, saving, or closing a form.

A rectangular button with a dark gray background and a thin black border. The word "Save" is centered in a white, sans-serif font.A rectangular button with a dark gray background and a thin black border. The word "Submit" is centered in a white, sans-serif font.

For detailed information about buttons, see “Buttons” on page 67.

**Button (Non-XForms)**

See **Button**.

**Button (Submit)**

Creates a submit button associated with an *xforms:submit* option.

**Button (Trigger)**

Creates a trigger button associated with an *xforms:trigger* option.

**Button (Upload)**

Creates a button associated with an *xforms:upload* option.

**Case** Creates a case associated with an *xforms:switch* option.

**Cell** Creates a choice within a combo box, list or popup. They are not themselves visible on the form, but do effect the appearance of lists to which they are linked.

**Cell (Non-XForms)**

See **Cell**.

**Check** Creates a check box. Use check boxes when you have options that are either true or false, or on or off. (The value option of this item type is always either on or off.)

**Check all that apply:**

- Newspaper
- Radio
- Television

For detailed information about check boxes, see “Creating check box lists” on page 76.

**Check (Input)**

Creates a check box associated with an *xforms:input* option.

**Check (Non-XForms)**

See **Check**.

**CheckGroup (Select)**

Creates a group of check boxes associated with an *xforms:select* option, allowing the user to pick more than one item in a list.

**CheckGroup (Select1)**

Creates a group of exclusive check boxes that allow users to pick only one item in the list. (*xforms:select1* option)

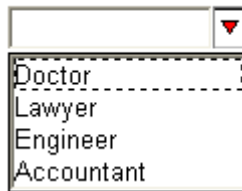
**Choice (Item)**

Creates a single *XForms:item* within a choice of combobox, list, popup, checkgroup or radiogroup.

**Combobox**

Creates a label and popup with a list of choices. The box is one or two rows in height, depending on whether it has a built-in label. After the box is selected, a list of choices “pops up” for the user. The user can select one of these choices, or type in a choice not presented in the list. Alternatively, the combo box can be configured to present the user with a calendar (See **Date Picker** item).

**Profession:**



For detailed information about combo boxes, see “Creating combo box lists” on page 78.

**Combobox (Non-XForms)**

See **Combobox**.

**Combobox (Select1)**

Creates a combobox associated with an *xforms:select1* option, allowing the user to pick only one item in the list or type in their own value.

**Custom item**

Creates items that are not part of standard XFDL. You define these items yourself, and put them in another namespace, such as Custom. They are never visible on the form but can be referenced by form computes and can contain custom options. Use custom items to integrate the form with other applications.

For detailed information on custom items, see the *Workplace Forms XFDL Specification* document.

### Date Picker

Creates an XFDL combo box to select a date, in which case the user can type in a date or select one from the calendar. If a built-in label is used, it is visible to the user at all times.

**Date:**



For detailed information about calendars, see “Creating calendars” on page 80.

### Date Picker (Input)

Creates a date picker associated with an *xforms:input* option.

### Date Picker (Non-XForms)

See **Date Picker**.

**Field** Creates a user input area on the form. Use fields to collect information from the user, such as names, dates, dollar amounts, and so on.

You can set up fields to check and restrict user’s entries, to flag errors and omissions and provide help on how to correct them, to format user input in a standard style, and to perform calculations and make logical decisions.

For detailed information about formatting and controlling input, see “Fields” on page 59.

**Home Address:**

10 Downing Street London, England
--------------------------------------

### Field (Non-XForms)

See **Field**.

### Field with Label

Creates a user input area on the form with a label. The label is relatively aligned to the left of the field. When you move the field, the label will also move with it.

For detailed information on relative alignment, see “Alignment types” on page 44.

### Field (Input)

Creates a single-line field associated with an *xforms:input* option. For detailed information about Field (Input), see “Creating single line fields” on page 140.

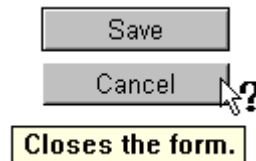
**Field (Secret)**

Creates a write-only field associated with an XForms secret. For detailed information about Field (Secret), see “Creating password fields” on page 141.

**Field (TextArea)**

Creates a multi-line field associated with an *xforms:textarea* option. For detailed information about Field (TextArea), see “Creating multi-line fields” on page 141.

**Help** Stores the context-sensitive help messages you add to your form. They are linked to specific items in the form, and the message they contain displays when the user asks for help with that item.



For detailed information about help, see “Help messages” on page 54.

**Label** Creates a field for text and images on the form. Use labels to display titles, instructions, logos, and other graphics.

**Label (Output)**

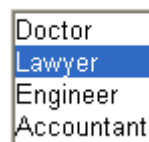
Creates a label associated with an *xforms:output* option.

**Label (Non-XForms)**

See **Label**.

**Line** Creates a line on your form. Typically you would use the line to separate items or to logically divide items on your page. They can be horizontal or vertical, and of any thickness. Lines with a thickness of four pixels or more will appear three-dimensional in the Viewer.

**List** Creates a predefined list of items to choose from. The user can select only

**Profession:**

one choice.

For detailed information about lists, see “Lists and choices” on page 75.

**List (Non-XForms)**

See **List**.

**List (Select)**

Creates a list associated with an *xforms:select* option, allowing the user to pick more than one item in the list. For detailed information about lists, see “XForms lists” on page 141.

**List (Select1)**

Creates a list associated with an *xforms:select1* option, allowing the user to pick only one item in the list. For detailed information about lists, see “XForms lists” on page 141.

**Page** Like paper pages, form pages provide the surface on which you place form items.

For detailed information on pages, see “Pages” on page 25.

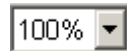
### Pane (Group)

Creates a pane associated with an *XForms:group* option. A pane with a group allows you to create a group of items that can be positioned or made visible as a unit, and that can be given a common border or background. (*xforms:group* option.)

### Pane (Switch)

Creates a pane associated with an *XForms:switch* option. A pane with a switch allows you to group items into *cases* and then display one case of items at a time to the user. A case creates a set or group of times used with a switch. For detailed information about Switches, see “XForms conditional items” on page 148.

**Popup** Creates a popup list of choices. The list is a single row in height until selected, at which point a list of choices “pops up” for the user. The user can then select one of the choices. If a built-in label is used, it will be visible only until the user selects a choice.



For detailed information about popups, see “Creating popup lists” on page 77.

### Popup (Non-XForms)

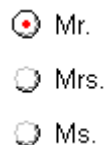
See **Popup**.

### Popup (Select1 )

Creates a popup list associated with an *xforms:select1* option, allowing the user to pick only one item in the list.

### Radio

Creates an XFDL radio button that presents mutually exclusive choices. The buttons are grouped together, and only one button from a group can be selected at a time. The buttons are linked in a group by setting the *group* option. Use radio buttons when you have a list of options, from which the user may choose only one. (The *value* option of this item type is always either on or off.)



For detailed information about radio buttons, see “Creating radio lists” on page 77.

### Radio (Non-XForms)

See **Radio**.

### RadioGroup (Select1)

Creates a group of radio buttons associated with an *xforms:select1* option, allowing the user to pick only one item in the list.

### Spacer

Creates space between items on a form. They are most useful when used with relative positioning. Because they are invisible when the form is displayed in the Viewer, you can use them to insert blank space between other items, or as invisible reference points for relative positioning. For detailed information about spacers, see “Spacing items” on page 45.



**Slider (Range)**

Creates a slider that offers users a range of values.

**Table (Repeat)**

Creates an XForms table into which you can place items organized into rows. For detailed information about tables, see “Creating XForms tables manually” on page 153.

**Table (Repeat) by Wizard**

Starts a wizard that creates an XForms table of rows and columns with specified display and configuration settings such as table lines, borders and row formatting. For detailed information about tables, see “Creating XForms tables using a wizard” on page 151.

**Toolbar**

Creates a distinct, fixed area at the top of a page where you can place headings, images, and control buttons. Toolbars appear at the top of forms when opened in the Viewer, but when the form is printed, the toolbar is omitted. For detailed information about toolbars, see “Toolbars” on page 57.

## Object Library items

You can add the following pre-made XFDL objects to your form:

**CDN Address Block**

Creates a Canadian address block containing a number of fields and a popup with some constraints applied. Three fields are required to be filled in by the user: *First Name*, *Last Name* and *S.I.N.* These fields are displayed in yellow and will change to white once the data has been entered. A number of the fields in this Canadian Address Block display error messages when the user does not type a specified pattern. Help tooltips are displayed, providing instructions on how to enter the correct data.

**Phone Number (###) ###-####**

Creates a phone number field with a label positioned to the left. The field uses a (...)...-.... template. An error is displayed when the user does not type the specified pattern. In addition, a help tooltip is displayed providing instructions on how to enter the correct data.

For detailed information on templates, see “Constraints types” on page 61.

**Phone Number with patterns**

Creates a phone number field with a label positioned to the left. This field checks and restrict the user’s entries to type a phone number in one of the following patterns:

- (###) ### - ####
- ###.###.####
- ###-###-####
- #####

An error is displayed when the user does not type the specified pattern. In addition, a help tooltip is displayed providing instruction on how to enter the correct data.

**Postal Code**

Creates a Postal Code field with a label positioned to the left. This field checks and restrict the user’s entries to type a A#A #A# pattern. An error is

displayed when the user does not type the specified pattern. In addition, a help tooltip is displayed providing instructions on how to enter the correct data.

### **Province Popup**

Creates a province popup list presenting the user with a list of provinces to choose from. The list is a single row in height until selected, at which point a list of choices “pops up” for the user. The user can then select one of the choices. The built-in label, *Select Province*, is visible only until the user selects a choice.

### **Province Popup (Abbreviated)**

Creates a province popup list presenting the user with a list of abbreviated names of provinces to choose from. The list is a single row in height until selected, at which point a list of choices “pops up” for the user. The user can then select one of the choices. The built-in label, *Select Province*, is visible only until the user selects a choice.

### **SIN (###-###-###)**

Creates a Social Insurance Number field with a label positioned to the left. This field checks and restrict the user’s entries to type a ###-###-### pattern. An error is displayed when the user does not type the specified pattern. In addition, a help tooltip is displayed providing instructions on how to enter the correct data.

### **SSN (#####-####)**

Creates a Social Security Number field with label positioned to the left. This field checks and restrict the user’s entries to type a #####-#### pattern. An error is displayed when the user does not type the specified pattern. In addition, a help tooltip is displayed providing instructions on how to enter the correct data.

### **State Popup**

Creates a United States popup list presenting the user with a list of states to choose from. The list is a single row in height until selected, at which point a list of choices “pops up” for the user. The user can then select one of the choices. The built-in label, *Select State*, is visible only until the user selects a choice.

### **State Popup (Abbreviated)**

Creates a United States popup list presenting the user with a list of abbreviated names of states to choose from. The list is a single row in height until selected, at which point a list of choices “pops up” for the user. The user can then select one of the choices. The built-in label, *Select State*, is visible only until the user selects a choice.

### **US Address Block**

Creates a United States address block containing a number of fields and a popup with some constraints applied. Three fields are required to be filled in by the user: *First Name*, *Last Name* and *S.S.N.* These fields are displayed in yellow and will change to white once the data has been entered. A number of the fields in this US Address Block display error messages when the user does not type a specified pattern. Help tooltips are displayed, providing instructions on how to enter the correct data.

### **Zip Code**

Creates an United States Zip Code field with label positioned to the left. This field checks and restrict the user’s entries to type either a ##### or #####-#### pattern. An error is displayed when the user does not type the

specified pattern. In addition, a help tooltip is displayed providing instructions on how to enter the correct data.

For detailed information on how to convert these items to XForms, see “Converting XFDL items to XForms” on page 49.

---

## Inserting items onto a page

You create the visible portion of a form by placing items from the Palette on the page.

To insert an item onto a page:

1. In the Palette, click a button.
2. Insert the item onto the canvas by either:
  - Moving the pointer over the canvas and clicking where you want to place the item. The default item size is inserted.
  - Moving the pointer over the canvas and click-dragging where you want to place the item. A re-sized item is inserted.

By default, when you do not specify the width and height 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, and so on. 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.

It is often easier to create items using the default sizing because they will all be created with the same dimensions. This will save you from adjusting the height or width later.

For detailed information about Palette buttons, see “Standard Library items” on page 29.

---

## Arranging items on a page

There are many tools that can help you position and arrange items on a page.

### Bounding boxes

Each item on the form has an invisible bounding box around it. Bounding boxes include all the elements of the item such as built-in labels and borders. Bounding boxes are not displayed in the Viewer, but you can view them in the Designer. This box surrounds the item, and determines the space that the item takes up on the form. This bounding box determines how items are positioned.


For example, if you place one item after another, those items are placed so that their bounding boxes are almost touching. If you align one item left-to-right with another, those items are placed so that their bounding boxes are touching.

This can be particularly important when using labels. The text or image a label displays may be smaller than the label’s actual size, and without a visible bounding box you might not be able to tell how big the label actually is.

To make the bounding boxes visible on your form:

1. Click **Window** → **Preferences**.
2. From the left column, expand **Workplace Forms** and **Designer View**.
3. In the Design View preferences, select the **Show Bounding Boxes around items** check box.

To set the size of a bounding box:

1. Select the item.
2. In the Properties view, click  and click **Show Advanced Properties**.
3. Expand **General** and **size**.
4. Click within the **size** value field.
5. Type the **width** and **height** values and press Enter. The bounding box automatically updates in the canvas.

**Note:** Bounding box sizes can only be changed for **Box, Button, Check, Combobox, Field, Label, Line, List, Popup, Radio, Slider (Range),** and **Spacer** items.

## Layout grids

The layout grid is an evenly spaced grid of horizontal and vertical lines that is superimposed on the form. This grid helps you line up items on the form and ensure uniform spacing.

### Showing or hiding the grid

To show or hide the grid:

Click **View** → **Show Grid**.

Once set on, grids are displayed in each working session.

### Changing the grid size

You can control the spacing between the lines of the grid, depending on how close together you want your items to be.

To change the size of the grid:

1. Click **Window** → **Preferences** to open the Preferences window.
2. From the left column, expand **Workplace Forms** and **Design View** to open the Design View preferences.
3. In the **Enter space between grid on X axis** field, type a new value.
4. In the **Enter space between grid on Y axis** field, type a new value.
5. Click **Apply**. The grid size automatically updates in the editor.
6. Click **OK**.

### Changing the grid color

To change the grid color:

1. Click **Window** → **Preferences** to open the Preferences window.
2. From the left column, expand **Workplace Forms, Design View** and **Colors**.
3. In the **Select Color for Grid** color box, click to select a new color. A Color window opens.

4. Select the color to use and click **OK**.

## Snapping to the grid

Snapping to the grid forces an item to align automatically to a point on the grid as you move the item. When you place an item on the form, it moves automatically so that its top left corner is placed on the nearest grid point.

You must drag items a minimum distance (at least three pixels) before the Designer repositions the item. This prevents accidental movement when double clicking the item.

### Note:

- When moving multiple items, the item in the top left of the group snaps to the grid, and all other items maintain their position relative to the top left item.
- When dragging the edge of an item to change its size, the item's edge will also snap to the grid.

To toggle grid snapping:

Click **View** → **Snap To Grid**.

## Snapping to items

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, allowing you to force an item to align to another item. When moving an object, additional guidelines will appear when its edge is in line with another item on the form. The edges that can be used as snapping points are the top, left, right, bottom, and center.

To toggle item snapping:

Click **View** → **Snap to Geometry**.

## Rulers and guides

Rulers and guides help you to measure the size and position of items on the form. Rulers and guides are useful for creating a clean, symmetrical layout for your form.

Rulers enable you to measure items in inches or pixels. Guides are thin red lines that you can place directly on the form to help you align items manually.

Rulers appear automatically on the top and left borders of the form when you create a new form.

### Showing or hiding rulers

Use the rulers at the top and left sides of the canvas to accurately place items.

To show or hide rulers:

Click **View** → **Show Rulers**.

Once set on, rulers are displayed in each working session.

## Creating guides

A guide is a horizontal or vertical reference line that spans the form. They appear as thin lines in the Designer, but are not visible when the form is opened in the Viewer. You set up guides to assist in aligning items exactly.

To create a guide:

1. Click **View** → **Show Rulers**.
2. Click on the ruler and drag the guide to the desired position.

**Note:** Guides are not saved after closing the file.

## Moving guides

To move a guide:

1. Click and hold on a guide marker. The guide displays horizontal or vertical arrows, depending on which ruler the guide is positioned in.
2. Release the mouse button when you have dragged the guide to the appropriate position.

## Deleting guides

To delete a guide:

1. Within the ruler area, click on the guide you want to delete. The guide's marker becomes highlighted.
2. Press Delete. You can also drag the guide marker off the ruler.

## Zooming and display

You can zoom your view of the canvas, show or hide page edges, and show or hide items.

### Zooming the canvas

Use the zoom function to zoom in or out of the form's page.

To zoom the canvas:

Do one of the following:

- Click **View** → **Zoom In**.
- Click **View** → **Zoom Out**.
- Select a zoom level from the toolbar.

### Showing or hiding page edges

You can show or hide lines within the canvas to indicate the edge of the page.

To show or hide page edges:

Click **View** → **Show Page Size**.

### Showing or hiding items

You can show or hide items on the canvas. You may want to hide items to work on elements of an item or items without disturbing the others on a page.

**Note:** This does not affect the visibility of the items in the final form.

To show or hide items:

1. In the Outline view, click the items you want to view on the canvas.
2. In the Outline view toolbar, click the **Filter on Visible** button. Only the items you have selected display on the canvas.

To show all items on the canvas, click **Filter on Visible**.

## Selecting items

To select an item, click on it. A bounding box displays around the item, indicating that you selected it. Alternatively, click the item name in the Outline view.

To select multiple items:

Do one of the following:

- Hold down the Ctrl key and click on each item.
- To select individual items in a group, use the **Marquee** tool in the Palette. Hold down the left mouse button and drag the pointer diagonally on the form to surround the items.

To select all of the items on the form at once, click **Edit** → **Select All**.

## Moving items

To move an item:

Do any of the following:

- Select the item and drag it. If you want the item to move in one direction only (vertically, horizontally or diagonally), hold down the Shift key while you drag it.
- To nudge an item by 1 pixel at a time, select the item and press an arrow key.
- To nudge an item by the grid spacing, select the item, hold down the Ctrl key and press an arrow key.
- In the Properties view, expand **General**, **itemlocation**, **Location List** and set the x and y values.

To undo a move, click **Edit** → **Undo Move Object**.

## Resizing items

To resize an item:

1. Select the item.
2. Do one of the following:
  - Click and drag its edges.
  - In the **Properties** view, expand **General**, **itemlocation**, and **Location List** and set the new values for x and y.
  - Nudge or resize items by 1 pixel at a time.
    - a. Select the item.
    - b. Press the period key on your keyboard. The cursor changes to a two way arrow on a side of the item.

- c. Move from one edge to another by pressing the period key on your keyboard.
- d. Once you have selected a side to resize, use the arrow keys on your keyboard to nudge the size of the item in 1 pixel increments.
- e. Press Enter to accept the new size of the item.

## Changing the build order of items

When you insert items onto a page, the order in which they are created plays a part in how you can position them. As you place items on the page, they form a sequence. The first item you place on the page is the first item in the sequence, the second item you place is the second item in the sequence, and so on. This sequence is called the *build order*, since it is the order in which you build the form. The build order seldom reflects the order in which the items actually appear on the form, since the items can be moved anywhere once they have been created.

When you use relative positioning, you must be aware of this build order. If you position an item, any item that you use as a reference must precede the item you place in the build order. For example, if you created a label, then a field, and then a button, you would be able to position the button in relation to either the field or the label. However, you would not be able to position the label in relation to either of the other items, because those items come after it in the build order.

The build order of items on a page is the same as the order that the items are listed in the Outline view.

To change the build order of items:

1. In the Outline view, select an item and drag it up or down in the list.
2. When the cursor is at the location in the list where you want the item, release the left mouse button.

---

## Tab order

Tab order is the order in which the user can move from one item to another item on a page using the Tab key. Filling in a form is much quicker and easier if the user only has to press Tab to jump to the next item in a logical sequence.

Tab order is also referred to as *next tab order*. *Previous tab order* is the order in which the user can move from item to item on a page by pressing Shift+Tab. Previous tab order is often the opposite of next tab order; however, it is possible to set up next tab order and previous tab order so they are unrelated to each other.

When you insert items onto a page, the Designer automatically defines the tab order as the order in which you insert the items. (In other words, the default tab order is the same as the build order. In addition, the default previous tab order is the opposite of the default next tab order.) Unless you plan your form very well, this tab order will rarely be ideal. However, you can modify the tab order at any time.

## Changing the tab order of items on a page

To change the tab order of items on a page:

1. Click **View** → **Show Next Tab Order**. Lines connect each item on the page with arrows indicating the tab order:



- Grey lines indicate the default tab order between items.
- Blue lines indicate a user-defined tab order between items.
- Red lines indicate an error.
- For advanced users: Light blue lines indicate the tab order is based on the **first** or **last** option of the item's parent, and not the **next** or **previous** option of the item.

The **Connection Creation** button is also displayed within the Palette.

2. Click **Connection Creation**.
3. Click the item that you want to be first in the tab order (that is, the item that will receive focus when the form is opened first).
4. Click the item that you want to be second in the tab order. A blue line connects the items, indicating the user-defined tab order between them.
5. Continue clicking pairs of items to define the tab order between them. By default, when you set up a user-defined tab order, the corresponding previous tab order is automatically set up.
  - If you want to modify tab order connections directly, click **Select** and click the line connecting the items to select the connection. Handles (black squares) appear at each end of the connection. Drag a handle from one item to another to modify the connection.
  - If you want to delete a tab order connection between two items, click **Select** and click the line connecting the items to select the connection. Then press **Delete**.
6. Click **View** → **Show Previous Tab Order**.
  - If you want change the tab order, repeat steps 2 through 5.
7. Click **View** → **Hide Tab Order** to hide the tab order lines. The **Connection Creation** button also no longer displayed within the Palette.

---

## Aligning items

You can align items so their edges or centers are lined up.

There are two types of align:

- *relative align* — Changes the position of items so they align with another item (the reference item) and anchors the items to the reference item. If you later move or resize the reference item, any items anchored to it will automatically move in order to maintain alignment with the reference item.
- *absolute align* — Changes the position of items so they align with another item (the reference item). The items will not automatically move if you later move or resize the reference item.

Use relative align when the size or position of items on your form may vary (for example, when your form is receiving data from a database or when sections of your form are created dynamically).

To align items:

1. Select the items to move.
2. Select the reference item (that is, the item that will not move and that the other items will align to). If you align items using relative align, the reference item must be *before* the aligned items within the build order.
3. Right-click and select an alignment type from **Absolute Expand** or **Relative Expand**.

## Alignment types

You can align items by using the following alignment types.

**Above** Moves the items above the reference item (leaving three pixels between each item) and aligns their left edges.

**After** Moves the items to the right of the reference item (leaving three pixels between each item) and aligns their top edges.

**Before**

Moves the items to the left of the reference item (leaving three pixels between each item) and aligns their top edges.

**Below** Moves the items below the reference item (leaving three pixels between each item) and aligns their top edges.

**Top to Bottom**

Moves the items so their top edges align with the bottom edge of the reference item.

**Top to Center**

Moves the items so their top edges align with the center of the reference item.

**Top to Top**

Moves the items so their top edges align with the top edge of the reference item.

**Left to Center**

Moves the items so their left edges align with the center of the reference item.

**Left to Left**

Moves the items so their left edges align with the left edge of the reference item.

**Left to Right**

Moves the items so their left edges align with the right edge of the reference item.

**Right to Center**

Moves the items so their right edges align with the center of the reference item.

**Right to Left**

Moves the items so their right edges align with the left edge of the reference item.

**Right to Right**

Moves the items so their right edges align with the right edge of the reference item.

**Bottom to Bottom**

Moves the items so their bottom edges align with the bottom edge of the reference item.

**Bottom to Center**

Moves the items so their bottom edges align with the center of the reference item.

**Bottom to Top**

Moves the items so their bottom edges align with the top edge of the reference item.

**Center to Bottom**

Moves the items so their centers align with the bottom edge of the reference item.

**Center to Left**

Moves the items so their centers align with the left edge of the reference item.

**Center to Right**

Moves the items so their centers align with the right edge of the reference item.

**Center to Top**

Moves the items so their centers align with the top edge of the reference item.

**Horizontally Between**

Moves the reference item horizontally so it is spaced equally between two other items on the form.

**Center to Center Horizontally**

Moves the items horizontally so their centers align with the center of the reference item.

**Vertically Between**

Moves the reference item vertically so it is spaced equally between two other items on the form.

**Center to Center Vertically**

Moves the items vertically so their centers align with the center of the reference item.

## Spacing items

Because relative positioning relies on using other items as points of reference, you will often need to include invisible points of reference in order to put space between your items or position your items exactly where you want them. You can use spacers to create these invisible points of reference and to create space between items on your form.

For example, suppose you had a field that was positioned below a label, and you wanted to put some space between the two items. All you would need to do is create a spacer, position it below the label, and then position the field below the spacer. You could then adjust the size of the spacer so that it kept the correct amount of space between the other two items.

To use a spacers as invisible reference points for relative positioning:

1. In the Palette, click **Spacer**.
2. Click on the page.
3. Place another item on the page.
4. Hold the Shift key down and select a spacer to use as an *reference*.
5. Right-click and click **Relative Align**.
6. Select the modifier to position the spacer item.

Click **Preview**. Once the form is open, notice that the spacer is invisible.

## Removing relative alignment assignment

Relative positioning works by assigning locators to each item. Essentially, locators are two or three words that tell the Viewer where to put the item. Each locator consists of a modifier and either one or two item tags. The modifier describes how the item should be placed, while the item tags give reference points.

Items can have any number of locators assigned to them. Often, there will be one to set the horizontal position of the item, and another to set the vertical position. When an item has several locators, the Designer applies them in the order they were created.

To remove relative positioning settings assigned to an item:

1. Select the item you have applied relative positioning to.
2. In the Properties view, expand **General**, **itemlocation**, and **Location List**. X and Y properties are listed as well as the assigned relative alignment choices. For example, if you choose **Relative Align Below**, the alignment type **Below** is listed under **Location List**.
3. In the value field adjacent to the alignment property, click . The alignment assignment is deleted.

---

## Expanding items

You can simultaneously resize and align items by *expanding* them. This is a quick way to resize items so that they line up with other items exactly. For example, you can expand a label to the size of a field by using a Right to Right expansion. This will lengthen the label until it is the same size as the field and align the right edge of the label with the right edge of the field.

There are two types of expand:

- *relative expand* — Changes the size of items so they align with another item (the reference item) and anchors the items to the reference item. If you later move or resize the reference item, any items anchored to it will automatically change size in order to maintain alignment with the reference item.
- *absolute expand* — Changes the size of items so they align with another item (the reference item). The items will not automatically change size if you later move or resize the reference item.

To expand items:

1. Select the items to expand.
2. Select the reference item (that is, the item that the other items expand to). If you expand items using relative expand, the reference item must be *before* the expanded items within the build order.
3. Right-click and select an expansion type from **Absolute Expand** or **Relative Expand**.

## Expansion types

You can simultaneously resize and align items using the following expansion types:

### **Make Same Size**

Resizes the items to the size of the reference item.

**Make Same Width**

Resizes the width of the items to match the width of the reference item.

**Make Same Height**

Expands/contracts the height of the items to the height of the reference item.

**Top to Bottom**

Expands/contracts the top edge of the items to the bottom edge of the reference item.

**Top to Center**

Expands/contracts the top edge of the items to the center of the reference item.

**Top to Top**

Expands/contracts the top edge of the items to the top edge of the reference item.

**Left to Center**

Expands/contracts the left edge of the items to the center of the reference item.

**Left to Left**

Expands/contracts the left edge of the items to the left edge of the reference item.

**Left to Right**

Expands/contracts the left edge of the items to the right edge of the reference item.

**Right to Center**

Expands/contracts the right edge of the items to the center of the reference item.

**Right to Left**

Expands/contracts the right edge of the items to the left edge of the reference item.

**Right to Right**

Expands/contracts the right edge of the items to the right edge of the reference item.

**Bottom to Bottom**

Expands/contracts the bottom edge of the items to the bottom edge of the reference item.

**Bottom to Center**

Expands/contracts the bottom edge of the items to the center of the reference item.

**Bottom to Top**

Expands/contracts the bottom edge of the items to the top edge of the reference item.

---

## Cutting, copying, pasting and deleting items

When designing your form, you can cut, copy and paste items to different locations on a page, to another page, or to another form.

You can also cut and paste items to change the build order.

## Cutting items

Cutting an item removes the item from the page and saves it to the clipboard.

To cut selected items:

Click **Edit** → **Cut**.

## Copying items

Copying an item saves the item to the clipboard.

To copy selected items:

Click **Edit** → **Copy**.

## Pasting items

Pasting an item inserts the contents of the clipboard into the form. When you paste an item, the Designer adds it to the build order immediately after the item that is currently selected. If the page global is selected (by clicking the background of the form), the item will be added to the end of the build order.

**Note:** When you copy a label or button using an image to another page in the form, the image is referenced to the original file.

To paste cut/copied items:

Click **Edit** → **Paste**. If you paste the item onto another page, the item is pasted in the same x and y location as the original item.

## Deleting items

Deleting an item removes the item from the page.

To delete selected items:

Click **Edit** → **Delete**. Alternatively, press Delete.

If you are deleting an item (or items) that are referenced by formulas in other items, a window opens asking whether you really want to delete the items. Click **Yes** if you want to delete the item referenced by the formula. Click **No** if you do not want to delete the referenced item, but do want to delete the other items you selected. Click **Cancel** to not delete anything.

---

## Visibility

Visibility properties allow you to set an item to be invisible in two scenarios: when the form is displayed in the Viewer or when the form is printed. This allows you to create hidden items on the form, or to strictly control which portions of the form are printed. Regardless of these settings, you can always see invisible items when working with a form in the Designer.

To set an item to be invisible:

1. Select the item.
2. In the Properties view, expand **Appearance**.
3. Set **visible** to **off**.

Tip: To hide the items in the Designer that have visible set to off, click **Window** → **Preferences** , expand **Workplace Forms** and **Design View**. Select the **Hide items that have their visible property turned off** check box.

---

## Converting XFDL items to XForms

Certain XFDL items can be converted into XForms items. The XForms items available will depend on the XFDL item that you want to convert. For example, an XFDL Field can be converted into either an XForms Input, TextArea or Secret, while an XFDL Label can only be converted into an XForms Output.

**Note:** You can only convert items to XForms items when XForms support is enabled. For detailed information about adding XForms support to an existing form, see “Adding XForms support to an existing form” on page 130.

To convert an XFDL item to XForms:

1. Select the XFDL item to convert.
2. Right-click and click **Convert Item** and then click the appropriate XForms item.





---

## Setting item properties




Each item in your form has various properties that control its appearance and behavior (for example, the font of a text label, the action triggered by a button, and so on). By setting item properties, you can control the appearance and behavior of items.

**Note:** What is referred to as a *property* in the Designer, is referred to as an *option* in XFDL.

To set item properties:

1. Select the item.



Tip: Press Shift to select several items and set their common properties to the same values.

2. In the Properties view, click within the field of the property you want to set.
3. Do either of the following:
  - For properties that accept text, type the value directly within the field, or click the  button within the field to open a text editor.
  - For properties that accept one of several choices (for example, on/off or left/right/center), click the  button within the field and select the setting.
  - For other properties (for example, **fontinfo** or **fontcolor**), click the  button within the field to open the editor for that type of property (for example, a font selection window or a color selection window).

For detailed information about specific properties, see “Appendix B: Options” on page 187.

---

## Resetting a property to its default value

When you select an item, its properties are displayed in the Properties view. Properties that are set to their default value are indicated by a  (white circle). Properties that have been modified from their default value are indicated by a  (blue circle).

To reset a property to its default value:

1. Select the item.
2. In the Properties view, click . The  changes to .

---

## Copying a property setting from one item to another

To copy a property setting from one item to another:

1. Select the item that contains the property setting you want to copy. The item’s properties are displayed in the Properties view.
2. In the Properties view, click within the value field of the property you want to copy.
3. Right-click and click **Copy**.
4. Select the item to copy the property setting *to*. The item’s properties are displayed in the Properties view.

5. In the Properties view, click within the value field of the property you want to replace.
6. Right-click and click **Paste**.

---

## Changing colors

You can use color to improve your form's appearance and to help the user fill them out.

Colors are defined by three components: Red, Green and Blue. Each component has a numerical value between 0 and 255. The exact color is determined by the number value of the separate components. White (which consists of the maximum values of every color) has a RGB value of 255, 255, 255. Black (which consists of no color) has an RGB value of 0, 0, 0. Pure red has an RGB value of 255, 0, 0.


You do not need to know the exact RGB value of the color you want to use. If you choose to use a custom color, you can select the color from the screen. Use the RGB values for fine-tuning the color you want.

You may wish to avoid using the colors that the Viewer uses to flag errors or mandatory items to users. These are the first two colors in the top row of the color picker:

Meaning	Color	RGB value
Mandatory	Light yellow	255, 255, 208
Error	Watermelon	255, 128, 128

## Changing the background and text color

To change the color of the background and text used in your form:

1. In the Outline view, click **Page Global**.
2. In the Properties view, expand **Appearance**.
3. Click within the **fontcolor** or **bgcolor** value field.
4. Click  and select the color you want to use.


**Note:** **backgroundimage** enables you to use an image as a template to create your form. The assigned image is not supported in print or preview.

For detailed information on these color properties, see the *Workplace Forms XFDL Specification* document.

## Changing the toolbar color

You can change the color of the form's toolbar to make it stand out from the rest of the form.

To change the color of a toolbar:

1. Click a blank area in the tool bar.
2. In the Properties view, expand **Appearance**.
3. Click  within the **bgcolor** value field.
4. Select a color and click **OK**.

## Changing the color of an item

You can change the color of backgrounds, text, and borders for both items and their built-in labels.


To change item colors:

1. Select the item.
2. In the Properties view, expand **Appearance**.

The following color properties are available:

- **bgcolor**
- **fontcolor**
- **labelfontcolor**
- **labelbgcolor**
- **printfontcolor**
- **printbgcolor**
- **printlabelfontcolor**
- **printlabelbgcolor**

**Note:** By default, the Properties view only shows the most commonly used properties. To see advanced properties, see “Showing advanced properties.” The **Miscellaneous** properties are then listed in the Properties view.

3. Click within the property value field.
4. Click  and select the color you want to use.

**Note:** If you set the **fontinfo** option using the Font properties window, setting Color or Strikethrough in the window will have no effect. To set the font color, use the **fontcolor** option. Strikethrough text is not supported by XFDL.


For detailed information about these color properties, see the *Workplace Forms XFDL Specification* document.

---

## Showing or hiding categories

By default, properties are grouped into categories (for example, General, Appearance, Format, and so on) within the Properties view. You can hide categories so properties are displayed as a single list.

To show or hide categories:


In the top-right corner of the Properties view, click  and click **Show Categories**.

---

## Showing advanced properties

By default, the Properties view only shows the most commonly used properties and does not show less commonly used, or *advanced*, properties.

To show advanced properties:

In the top-right corner of the Properties view, click  and click **Show Advanced Properties**.

---

## Sorting properties in alphabetical order

By default, the Properties view displays the most commonly used properties at the top of the list.

To sort properties in alphabetical order:

In the top-right corner of the Properties view, click  and click **Sort Alphabetically**.

---

## Help messages

There are three kinds of help messages that you can set up for the users of your forms:

- context-sensitive help messages
- error messages
- accessibility (audio help) messages.

Context-sensitive help messages are displayed when the user enters Help mode in the Viewer. While in Help mode, the user can point to items on the form and receive help messages you have written for those items. Each item on the form can have its own help message.

Error messages are displayed when a user's entry in a field or list does not match the required formatting. These messages are best used to indicate which format to use when entering information into a specific field. Each field and combobox on the form can have its own error message.

Accessibility messages are played by screen reading software to describe each item on a form to visually impaired users. The text entered in this option will not be displayed in the Viewer. When the focus moves to the item, an active screen reader will read this message. Note that screen readers vary in behavior, and may provide additional information about the item.

## Adding context-sensitive help to an item

Context-sensitive help messages are displayed when the user enters Help mode in the Viewer. You can add a help message to the page and then assign it to one or more items.

To add context-sensitive help to an item:

1. In the Palette, click **Help**.
2. Click on the canvas.  
A help item is created and listed in the Outline view.
3. In the Outline view, click the help item you created (for example HELP1).
4. In the Properties view, expand **General**.
5. Click within the **value** field and type a help message.
6. Select the items you want the help message applied to.

7. In the Properties view, expand **Help**.
8. Within the **Help** value field, type the name of the help item (for example, *HELP1*).

The Viewer will display the help message whenever the user asks for help with that form item.

Tip: To see the help messages you create, users must click the button in the Viewer toolbar with the arrow and question mark. You may want to remind your users of this by placing a note somewhere on your form.

## Creating an error message for a field or combo box

Error messages are displayed when a user's entry in a field or list does not match the required data. Use these messages to indicate the format and type of data the user should enter. Each field and list on the form can have its own error message. If you do not specify an error message, the user will see the default error message, which says, *This entry is invalid. Please try again.*

To create an error message for a Field or Combobox:


1. Select the field or combo box which you want to create an error message.
2. In the Properties view, expand **Format**, **format**, and **constraints**.
3. Click within the **message** value field.
4. Type the error message in the message field.

Tip: The default error message for a field or combo box is: "This entry is incorrect. Please try again." You can use this if you do not want to customize the error message.

## Adding accessibility messages

This accessibility property lets you include a description of the item that will be used by screen reading software. The text entered in this option will not be displayed in the Viewer. When the focus moves to the item, an active screen reader will read this message. Note that screen readers vary in behavior, and may provide additional information about the item. If you are concerned about accessibility, be sure to test your form with the appropriate screen reading software.

To add an accessibility message:

1. Select the item.
2. In the Properties view, expand **Help**.
3. Click within the **acclabel** value field.
4. Click  to open the editor.
5. Type the message you would like the screen reader software to read aloud.  
This message should contain additional information about the item to assist users with vision impairments. For example, *Mandatory field. Type your last name, first name, and middle initial.*
6. Click **OK**.

For detailed information about creating accessibility messages, see the *Workplace Forms Best Practices for Form Design* document.



---

## Toolbars

Toolbars allow you to place headings and control buttons on your forms so they are always visible for users. Toolbars appear at the top of forms when they are opened in the Viewer, but when the form is printed, the toolbar is omitted. The toolbar is not part of the printed area of the form, and can therefore be used for items which do not need to be printed, such as control buttons.

You can add a toolbar to any page of your form. This toolbar creates an area at the top of the page that will not scroll with the rest of the form, and will always be visible to the user. This is useful if you have a number of buttons that you always want the user to have access to, such as save or submit buttons, or if you have a title that you always want to be visible.

### Tips on setting up your Toolbar:

- Create a Toolbar with all the necessary buttons, images and text you need then duplicate this toolbar to paste onto each of the additional pages.
- Make sure your buttons are set up properly before you start working with them. Decide which buttons you will need, such as a Print button, a Save button, a Submit button, and so on. Then use one button as the basis for aligning all the others relative to one another on the form.

For detailed information see, “Buttons” on page 67, “Images” on page 81 and “Aligning items” on page 43.

---

## Adding a toolbar to a page

To add a toolbar to the current page:

1. In the Palette, click **Toolbar**.
2. Click on the canvas. A toolbar is placed at the top of the page.

Tip: When you create a toolbar, the Designer gives it a **sid** of TOOLBAR and makes it the first item in the build order. Do not change the toolbar’s **sid** or location in the build order. Toolbars must be the first item in the build order.

---

## Adding items to a toolbar

To add items to a toolbar:

1. In the Palette, click an item.
2. Click within the toolbar area of the form.

**Note:** The Designer automatically resizes the toolbar so that the items fit.

---

## Resizing a toolbar

The Designer automatically resizes the toolbar to fit the items you place in the toolbar.

If you want to increase the size of the toolbar, simply move one of the items in the toolbar down, so that part of it is past the lower edge of the toolbar. The Designer will automatically adjust the size of the toolbar to fit the item.

Alternatively, click the baseline of the toolbar and drag it down to a new position. The cursor changes to an up and down arrow icon. Release the mouse to set the size.

---

## Copying a toolbar from one page to another

You can copy a toolbar, including all of the items in the toolbar, to another page. This is useful if you want to use the same toolbar on each page of your form.

To copy a toolbar to another page:

1. In the Outline view, expand the page containing the toolbar you want to copy.
2. Select the toolbar.
3. Right-click and click **Copy**.
4. Select the page you want to copy this toolbar to.
5. Right-click and click **Paste**.



---

## Fields

A field is an area on the form where the user can type in information. Each field has a built-in label that will display text. Fields can also be configured to perform edit checks and to automatically format user input.


### When to use fields:

- Use fields to collect information from the user, such as names, dates, dollar amounts, and so on. You can set up fields to check and restrict user's entries, to flag errors and omissions and provide help on how to correct them, to format user input in a standard style, and to perform calculations and make logical decisions.
- To be efficient, your form should contain and display accurate data in standardized formats. You can achieve this by controlling and formatting field input.

---

## Creating a field

To create a field:


1. In the Palette, click **Field**.
2. Click on the canvas to insert a default size button, or drag in the canvas to insert a field of any size.
3. In the Properties view, click  and click **Show Advanced Properties**.
4. Set the properties for the field.

---

## Creating a field with a label

The pre-made object, **Field with Label**, creates a user input area on the form with a label relatively aligned to the left of the field.

To create a field with a label:

1. In the Palette, click **Field with Label**.
2. Click on the canvas to insert a default size button, or drag in the canvas to insert a label of any size.
3. In the Properties view, click  and click **Show Advanced Properties**.
4. Set the properties for the field and label.

---

## Specifying the type of data to accept

The default datatype for a field is **string** which allows the user to enter both characters and numbers.

You can set up fields to accept only a specific kind of input, such as dates or dollar values. If a user attempts to input data that the item does not accept, an error message will appear and the field will turn red.

To specify the type of data to accept:

1. Select the field.
2. In the Properties view, expand **Format**, and **format**.
3. Set **datatype** to the type of input you want to apply.

For a list of the types of input you may specify, see “Data types.”

## Data types

The types of data input you may specify are:

### **date\_time**

A date and time. For example, May 3, 2006 13:30 or May 3, 2006 1:30 PM.

### **currency**

A number that includes up to two decimal places. For example, \$30.05 or \$25.02.

The default US currency symbol, \$, is inserted before the number entered.

**float** A floating point number. For example, 134.0095.

### **integer**

A whole number, including zero and positive and negative numbers.

**month** A month.

The user can type three or more characters or a number between 1-12.

**string** A text field.

The user can type characters and numbers.

**time** A time. For example, 13:30 or 1:30 PM.

The user can type characters and numbers.

**void** Disables the entire format option (including data type, presentation, and constraints)

The user can type characters and numbers.

**year** A year, represented by 4 digits.

The user can type 2 or 4 digits.

**date** A date. For example, 5 May 2006.

### **day\_of\_week**

A day of the week. For example, Monday

The user can type a number between 1-7 or three characters to represent the day of the week.

### **day\_of\_month**

A day of the month. For example, 18.

The user can type 1-31.

For detailed information about presentation properties, see the *Workplace Forms Locale Specification for XFDL* document.

---

## Specifying the constraints on data

You can set up XFDL input items to accept only input formatted within certain constraints.

For example, you can control how users enter their Postal/Zip code information. In this scenario, setting **constraints** properties, such as setting **patterns** to @## #@#, #####, #####-#####, and setting the **presentation** property **casetype** to **upper**, controls the type of data the user can enter. If done incorrectly, the user is asked to re-enter the data.

To specify constraints on user input:

1. Select the item.
2. In the Properties view, expand **Format, format, and constraints**.
3. Set the constraints for the field.

For a list about the constraints you may apply to items, see “Constraints types”

For detailed information about constraints and presentation formatting properties, see the *Workplace Forms Locale Specification for XFDL* document.

### Constraints types

#### checks

Allows you to force the format check to fail, or to ignore all constraints settings. Valid settings are:

**fail** — Forces the format check to fail.

**ignore** — Ignores all constraint settings. Note that the data type and the presentation settings are still respected.

**none** — Has no effect.

Default: none.

#### mandatory

The user is required to fill in data before proceeding.

**range** The entry must fall within the range you specify. This range can be alphabetical or numeric. For example, if you set a range of Adams to Lee, an entry of McAllister would fall outside the range you specified and the input item would not accept it.

**length** The number of characters in the entry must fall within the limits you specify. For example, if you specified a length limit of five to seven characters, an entry of three characters would not be acceptable.

When you specify a length, take into consideration any extra characters that formatting, if specified, may add to the entry. When the input item counts the number of characters in an entry, it includes the characters added by formatting. For example, if you specified that a field should add a dollar sign to the beginning of the entry, the field will consider the dollar sign a character. Therefore, if you specify that the field should accept only four characters, the user will be able to type only three, unless the first one is a dollar sign.

#### patterns

Allows you to set one or more patterns for strings, dates, or numbers that

are valid as input. For example, you might want to constrain dates to the following format: YYYY-MM-DD. You must define each pattern in its own tag.

Note that the pattern setting overrides both the **style** and **negativeindicator** settings.

For detailed information about templates and patterns formatting, see the *Workplace Forms Best Practices for Form Design* and the *Workplace Forms Locale Specification for XFDL* documents.

#### **message**

Sets the message that is displayed when the input is invalid. This can be any text.

The default is *This entry is invalid. Please try again.*

#### **template**

Allows you to display symbols in the input area before the user enters their data. This is useful if you want to show formatting placeholders, such as parentheses for the area code in a phone number.

To create a template, use a period to represent any 1 character that the user types in. All other characters are shown to the user as typed.

For example, if you create the following template: (...)...-.....

The user will see the following: ( ) -

Setting a template in no way limits the user input. If you want to limit the user input, you must also use the patterns setting. Furthermore, you can only set one template for each item.

#### **groupingseparator**

Defines one or more symbols that are allowed to separate groups of numbers (such as thousands in North America) during input. This is often a comma, as shown: 1,000,000.

#### **decimalseparator**

Defines one or more symbols that are allowed to indicate the decimal place. In North America this is often a period, as shown: 100.00.

## **Setting up a mandatory field**

If you want to make sure users fill in a particular item, you can specify it as mandatory.

When the user opens a form, all mandatory input items will appear with a background of a pre-specified color, to signal to the user that they must be filled in. If the user tries to tab past a mandatory input item, an error message will appear. The default error message warns the user *This entry is invalid. Please try again.* If the user tries to submit the form without filling in a mandatory input item, the following warning is displayed: *One or more of the items in this form contains an invalid value. Do you wish to proceed anyway?*

To set an input item as mandatory:

1. Select the item.
2. In the Properties view, expand **Format, format, and constraints**.
3. Set **mandatory** to on.

For detailed information about how to change a help message, see “Creating an error message for a field or combo box” on page 55.

---

## Reformatting input data

You can set up fields to reformat data that is entered into them. This helps you to ensure consistency in the data that is submitted for processing. For example, you can set up a date field to convert all dates to numeric format before the form is submitted, regardless of the format in which the user originally entered them. Or you can set up a label to convert all dates to long format before the form is displayed, regardless of how the information appears in a database.

To reformat input:

1. Select the item.
2. In the Properties view, expand **Format**, and **format**.
3. Click within the **datatype** value field.
4. Set **datatype** to the type of data the field will accept. Depending on the type of data you set the input item to accept, the Designer will let you choose different formatting **presentation** options.
5. Set the **presentation** properties to apply.

For detailed information about presentation properties, see the *Workplace Forms Best Practices for Form Design* and the *Workplace Forms Locale Specification for XFDL* documents.

## Presentation types

The formats you may apply are the following.

For detailed information about presentation properties, see the *Workplace Forms Best Practices for Form Design* and the *Workplace Forms Locale Specification for XFDL* documents.

### Strings

For a **datatype** set to **string**, set **casetype** to one of the following:

**lower** Converts the entry to lower case characters.

**upper** Converts the entry to upper case characters.

**title** Converts the first letter of each word in the entry to an upper case character.

### Numbers and currency

For a **datatype** set to **integer**, **float**, or **currency**, set any of the following:

#### **groupingseparator**

The symbols used to separate groups of numbers (for example, thousands in North America). This is often a comma, as shown: 1,000,000

You can use any string, with the keyword **none** representing no separators at all. However, you should not use strings that already have a meaning, such as a period.

The default for the en\_US locale is a comma.

#### **negativeindicator**

Sets the symbols that are used to indicate a negative value. You can place

symbols both before and after the number by setting a prefix and a suffix. To do this, you must set the **prefix** and **suffix** values in the Properties view.

The prefix and suffix are defined as strings. For example, if you set the prefix to an open bracket and the suffix to a close bracket, you will get a bracket negative. The following shows the bracket notation for negative 100: (100)

You can also leave either the prefix or suffix blank, so long as the other setting has a value.

Note:

- The **pattern** setting overrides the **negativeindicator** setting.
- Do not use this setting with currency data types.
- Do not use symbols that already have meanings, such as a period.

The default for the en\_US locale is a minus sign (-).

**round** Determines how values are rounded. Valid settings are:

**floor** — Always rounds down. For example, 46.9 becomes 46.

**ceiling** — Always rounds up. For example, 46.1 becomes 47.

**up** — Rounds values greater than 5 up, and values less than 5 down. For values equal to 5, it rounds up. For example, 46.5 becomes 47, while 46.4 becomes 46.

**down** — Rounds values greater than 5 up, and values less than 5 down. For values equal to 5, it rounds down. For example, 46.5 becomes 46, while 46.6 becomes 47.

**half\_even** — Rounds values greater than 5 up, and values less than 5 down. For values equal to 5, rounds up if the preceding digit is even, and down if the preceding digit is odd. For example, 46.5 becomes 47, while 45.5 becomes 45.

Note that if the **significantdigits** setting is used, then the round setting is reset to **half\_even**.

The default is **half\_even**.

**pad** Sets the number of digits to show, regardless of the value. For example, setting a pad of 5 would result in all numbers having five digits, as shown:

00002

00100

If the value has more characters than dictated by the pad setting, the value is not changed and is displayed as entered.

Use the padcharacter setting to control which character is used to pad the value.

The default is 0 (no padding imposed).

**padcharacter**

Sets the character to use for padding. For example, if you set the padcharacter to a zero and the pad setting was 5, numbers would be displayed as follows:

00010

01245

You may only specify a single character as the pad character. Furthermore, you must use a pad character that is valid for your data type. For example, you cannot use a Z in an integer value.

Use the pad setting to control how many pad characters are used.

The default for the en\_US locale is 0.

For detailed information about defaults for other locales, see the *Workplace Forms Locale Specification for XFDL* document.

#### **fractiondigits**

Sets the number of digits shown after the decimal place. For example, a setting of 3 would allow three digits after the decimal place, as shown:

13.764

All values are rounded according to the round setting. If no **round** setting is specified, all values are rounded up. (See the **round** setting for an explanation of rounding up.)

**fractiondigits** is only valid for float and currency data types.

**Note:** Setting both **fractiondigits** and **significantdigits** may cause conflicting formats. In this case, **significantdigits** takes precedence.

#### **significantdigits**

Sets the number of significant digits allowed. This is generally the total number of digits allowed in the number. For example, 134.56 has five significant digits.

If the data entered exceeds the number of significant digits allowed, then only the least significant digits are shown. For example, if you allow five significant digits and 12,345.56 is entered, then only 345.56 is shown.

**Note:** Setting both **fractiondigits** and **significantdigits** may cause conflicting formats. In this case, **significantdigits** takes precedence.

#### **pattern**

Allows you to set a pattern for number and date data types. This pattern is used to display the data. For example, you might want all numbers to be formatted with two digits after the decimal place.

**Note:** Pattern setting overrides both the style and **negativeindicator** settings.

#### **decimalseparator**

Defines one or more symbols that are allowed to indicate the decimal place. This is often a period, as shown: 100.00

You can use any string, such as a comma or a comma followed by a space.

**Note:** The user must use the same separator in a given string. For example, if you define both comma and space as valid separators, the user must type either 1,000,000 or 1 000 000. Mixing the separators, as in 1,000 000, is not allowed.

If this setting is empty, it inherits the **decimalseparator** defined in the presentation settings.

The default: is a comma.

## Dates

For a **datatype** set to **date\_time**, **month**, **year**, **date**, **day\_of\_week**, or **day\_of\_month** set **style** to:

**numeric**

Converts dates to numeric format. For example 20060613 (yyyyMMdd).

**short** Converts dates to short format. For example 2006-06-13 (yyyy-MM-dd).

**medium**

Converts dates to medium format. For example, 13 Jun 2006 (d MMM yyyy).

**long** Converts dates to long format. For example, June 13, 2006 (MMMM d, yyyy).

**full** Converts the date to full format. For example, Tuesday, June 13, 2006 (EEEE, MMMM d, yyyy).

---

## Changing scroll bars

When you add a field to your form, it allows users to enter more text than fits in the visible field. You can set up fields to show scroll bars, wrap text, and restrict users' entries to the visible field. The options you may use are:

- **scrollhoriz** — This adds a horizontal scroll bar to the field, and allows the user to make the text in the field wider than what is visible. To type a new line, the user must press Enter. This option allows you to set the scroll bar to wrap text to the next line if it is longer than the visible width of the field. The following options are available:
  - never** — permit scrolling using the cursor but display no horizontal scroll bar
  - always** — permit scrolling and display a horizontal scroll bar
  - wordwrap** — wrap field contents from line to line, inhibit scrolling and display no horizontal scroll bar
- **scrollvert** — This adds a vertical scroll bar to the field, and allows the user to type lines of text past the bottom of the visible field. The following settings are available:
  - never** — permit scrolling using the cursor but display no vertical scroll bar
  - always** — permit scrolling and display a vertical scroll bar
  - fixed** — inhibit scrolling and display no vertical scroll bars

To set up scrolling and input restrictions:

1. Select the field.
2. In the Properties view, expand **General**.
3. Set **scrollhoriz** to the setting you want.
4. Set **scrollvert** to the setting you want. For detailed information on scroll options, see the *Workplace Forms XFDL Specification* document.



---

## Buttons and actions

A form often contains buttons that let the user trigger actions (for example, saving the form). A form can also contain automatic actions that occur without the user explicitly triggering them (for example, submitting data to a database or server every five minutes).

---

### Buttons

Buttons are the most common way for users to trigger actions. You can also use cells within a popup list, combo box list, or box list to trigger actions. (For detailed information about presenting choices, see “Lists and choices” on page 75.)

You can use buttons or cells to let users perform the following actions.

- cancel** Closes the form. If any changes were made to the form since the last save or submit, then the user is told the form has changed, and is allowed to stop the cancellation.
- display** Allows the user to view one or more attached files. For detailed information about creating a display button, see “Creating attachment buttons” on page 106.
- done** Submits form data and then closes the form.
- enclose** Allows the user to attach one or more files in the form. For detailed information about creating an enclose button, see “Creating attachment buttons” on page 106.
- extract** Allows the user to extract a copy of one or more attachments to disk. For detailed information about creating an extract button, see “Creating attachment buttons” on page 106.
- link** Opens a file from the Internet or the user’s computer and displays it in a new window.
- pagedone** Switches to a different page in the form.
- print** Prints the page or the form, depending on the controls you set up.
- refresh** Refreshes the form. You may want users to refresh a form when viewing the form via Webform Server.
- remove** Allows the user to remove one or more attachments from the form. For detailed information about creating a remove button, see “Creating attachment buttons” on page 106.
- replace** Opens a file from the Internet or the user’s computer and displays it in the current window.
- saveform** Saves the form to the current file.

**saveas** Saves the form, prompting the user for a filename and location.

**select** For buttons, lets you capture an event in order to run a formula (compute). For example, you can setup a button that, when clicked, calls the duplicate function to duplicate a row of items.

For cells in a popup list, combo box list, or box list, records the user's selection.

This is the default **type** for buttons and cells.

**signature**

For buttons only: allows the user to sign the form. For detailed information about signatures, see "Signatures" on page 107

**submit**

Submits form data to a server and leaves the form open.


For detailed information on button types, see the **type** option in the *Workplace Forms XFDL Specification* document.

## Creating buttons

For detailed information on creating specific types of buttons, see:

- "Providing navigation between pages" on page 28.
- "Creating attachment buttons" on page 106.
- "Signatures" on page 107.

To create a button:

1. In the Palette, click **Button**.
2. Click on the canvas to insert a default size button, or drag on the canvas to insert a button of any size.
3. In the Properties view, expand **General**.
4. In the **value** value field, type the text that you want to appear on the button. (To display an image on the button, see "Adding an image to a button or label" on page 82.)
5. In the **type** value field, click the action type you want to use. For detailed information about button types, see "Buttons" on page 67).
6. In the Properties view, click  and click **Show Advanced Properties**.
7. Set the properties for the button.

## Creating submit buttons

Submit buttons let the user submit their completed form by HTTP or e-mail. If you are working in a server environment, you will probably want to use HTTP routing. E-mail routing is most useful in environments where there is no server available, or where the people using the form will not have access to Internet submissions.

There are two ways a user can submit a form via e-mail:

- **Using the Mail Form button in the Viewer toolbar** — If the user clicks the **Mail Form** button in the Viewer toolbar, a blank address form is displayed. The user can fill out the address form (for example, recipient, subject, cc, bcc, and message) and send the e-mail. The transmitted form is sent as an attachment to

this e-mail. Before sending the e-mail, the user can add further attachments, remove any attachments, or save any attachments to disk.


- **Using a submit button within the form** — You can setup a submit button within your form that acts similar to the **Mail Form** button in the Viewer toolbar. However, you can configure the button so that it automatically fills out some or all of the address form. (For example, you could configure the button so that it opens the address form and automatically addresses it to the correct person.)

You can setup a submit button to submit the form as an uncompressed XFDL file, a compressed XFDL file, or an HTML file.

Compressed XFDL files are compressed using a modified gzip format; this format is unique to Workplace Forms. You cannot view a compressed file in a text editor, but you can use the form in the Viewer and the Designer. Other compression software may not be able to decompress the forms.

When you submit a form in HTML format, only items that take user input and custom items are transmitted. Once a form is in HTML format, you cannot use the form in the Viewer or the Designer.

To create a submit button:

1. Select the button on the form.
2. In the Properties view, expand **General**.
3. In the **type** value field, click one of the following:
  - **submit** — to submit the form and keep the form open after submission
  - **done** — to submit the form and close the form after submission.
4. In the **url** value field, type the URL or e-mail address to submit the form to, for example:
  - a URL with the format: **scheme://host.domain[:port]/path/filename** for files and applications (where scheme is **http** or **https**)
  - a URL with a **mailto:** format:  
`mailto:user@host.domain?cc=user2@host.domain&bcc=user3@host.domain&subject=Timesheet&body=Form+is+attached`
5. Click  and click **Show Advanced Properties**.
6. If you want to specify the format used when the form is transmitted, expand **Transmit**.
7. In the **transmitformat** value field, click one of the following:
  - **application/vnd.xfdl** — Transmits the form as an uncompressed XFDL file
  - **application/vnd.xfdl;content-encoding="base64-gzip"** — Transmits the form as a compressed XFDL file
  - **application/x-www-form-urlencoded** — Transmits the form as an HTML file.

Sometimes it is useful for the form to be programmed to decide where it should go next. For instance, in a workflow the form may need to be signed by several people. Dynamically determining the recipient of the form via e-mail ensures the smooth processing of the workflow. For example, suppose a form needs to be filled in by Person 1, approved by Person 2, and finally received by Person 3. The logic goes: If Person 1 is using the form, then submit it to Person 2; if Person 2 is using the form, then submit it to Person 3. In this situation, you would set **url** to the following:

```
return address_field.value == 'Person1@acme.com' ?  
'mailto:Person2@acme.com' : 'mailto:Person3@acme.com'
```

## Filtering submissions

Filtering is an advanced feature that allows you to control which parts of a form are sent when a transmission occurs. For example, you can filter all of the buttons out of a form, or filter out everything but the buttons.

If you only have a basic knowledge of XFDL, or if you simply want to reduce the size of your form, you should use compression rather than filtering. However, if you need to reduce the size of your form further, or if the application that will receive the transmission does not need all of the form information, you may want to use filtering.

Filtering allows you to:


- Filter out all occurrences of an item type, such as all buttons or fields.
- Filter out groups of items or **datagroups**.
- Filter out all occurrences of a specific property, such as all background colors or active settings.
- Filter out individual items.
- Filter out all elements in a particular namespace, such as the custom namespace.

You can set up filters on any user control (such as a button or list choice) that causes the form to be transmitted, or in an automatic action that transmits the form.

There are two ways you can control what the submission button will and will not submit:

- You can specify the items that the button *will* submit. This is referred to as *keep* filtering.
- You can specify the items that the button *will not* submit. This is referred to as *omit* filtering.

To set up filtering for a submit button:

1. Select the button.
2. In the Property view, expand **Transmit**, and **transmitdatagroup**.
3. In the **filter** value field, click one of the following:
  - **keep** — lets you specify which items to submit.
  - **omit** — lets you specify which items to filter.
4. In the **Refs** value field, click  to add a **datagroupref** property.
5. In the **datagroupref** value field, type the name of the object to keep or omit. For example :
  - an item
  - a property (option)
  - a group of items
  - a datagroup
  - a namespace.
6. Continue adding **datagroupref** properties to keep or omit other objects.

## Creating link or replace buttons

To create a link or replace button:

1. Select the button on the form.
2. In the Properties view, expand **General**.
3. In the **type** value field, click one of the following:
  - **link** — lets the user open a file from the Internet or the user's computer and display it a new window.
  - **replace** — lets the user open a file from the Internet or the user's computer and display it in the current window.
4. In the **url** value field, type the URL for the file to link or replace.


## Creating save or cancel buttons

To create a save or cancel button:

1. Select the button on the form.
2. In the Properties view, expand **General**.
3. In the **type** value field, click one of the following:
  - **saveform** — lets the user save the form to the current file.
  - **saveas** — lets the user save the form and specify a filename and location for it.
  - **cancel** — lets the user close the form.

## Creating print buttons

To create a print button:

1. Select the button on the form.
2. In the Properties view, expand **General**.
3. In the **type** value field, click **print**.
4. Click  and click **Show Advanced Properties**.
5. In the Properties view, expand **Appearance**.
6. In the **printsettings** value field, set the properties. For more details about print settings, see *Workplace Forms XFDL Specification*.

---

## Automatic actions

Automatic actions allow you to set your form to perform an action automatically, such as submitting, without the user initiating the action. For example, you could set your form to refresh a field from a database every five minutes.

Actions can be set to occur once after a set period of time, or to repeat themselves continually. In general, the user has no control over these actions, and can only prevent them from occurring by closing the form.

Actions are useful whenever you want to set up a behind-the-scenes process. For example, you can use actions to have your form automatically open a file after five minutes, or continually send information to a database or a server.

The following automatic actions are available:

**cancel** Closes the form. If any changes were made to the form since the last save or submit, then the user is told the form has changed, and is allowed to stop the cancellation.

- display** Displays one or more attached files.
- done** Submits form data and then closes the form.
- link** Opens a file from the Internet or the user's computer and displays it in a new window.
- pagedone** Switches to a different page in the form.
- print** Prints the page or the form, depending on the controls you set up.
- refresh** Refreshes the form. You may want to refresh a form when deployed via Webform Server.
- replace** Opens a file from the Internet or the user's computer and displays it in the current window.
- saveform** Saves the form to the current file.
- saveas** Saves the form, prompting the user for a filename and location.
- select** The action item's active option goes from off to on to off again.
- submit** Submits form data to a server and leaves the form open.

You can set automatic actions to occur once after a set amount of time, or to repeat after a regular interval.

You will also have to set the delay interval. This is the number of seconds that should pass before the action occurs, or between repetitions of the action. Be aware that the count for the delay begins slightly before the form is actually visible on the screen, and that you should test your delays to be sure they are correct.

**Note:** Actions will only occur if they are on the page of the form currently open in the Viewer. Any actions on other pages will not occur until the user changes to the appropriate page, and then only after the specified delay has passed.

## Creating automatic actions

To create an automatic action:

1. In the Palette, click **Action**.
2. Do either of the following:
  - In the Outline view, click on the page that you want to insert the action onto.
  - If the page that you want to insert the action onto is displayed in the canvas, click on the canvas.
3. In the Properties view, expand **General**.
4. In the **type** value field, click the type of action you want to apply. For detailed information on action types, see "Automatic actions" on page 71.
5. In the Properties view, expand **delay** and **interval**.
6. In the **type** value field, click one of the following:
  - **once** — The action will occur once, after a delay interval you specify.
  - **repeatedly** — The action will occur repeatedly, waiting for an interval you specify between repetitions.

7. In the **interval** value field, type a number for the frequency (measured in seconds) of the repeated action (for repeating actions) or the delay before performing the action (for actions occurring once).





---

## Lists and choices

There are several ways of presenting a list of choices to users. You can also use a list of choices to let users trigger actions (for example, attaching a file to a form).

- **Check boxes**

**Check all that apply:**

Newspaper

Radio

Television

Use check boxes if you want the user to be able to select more than one of the choices you present. For example, you may want the user to “check all that apply” or present a single choice that the user can activate or deactivate.

- **Radio buttons**

Mr.

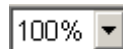
Mrs.

Ms.

Use radio buttons if you:

- Want the user to select only one choice.
- Want to show all of the possible choices at once.
- Want the user to have to click just once to select the choice.
- Have a small number of choices.

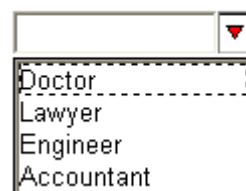
- **Popups**



A popup list appears as a single row on your form. It allows the user to select a single choice.

- **Combo boxes**

**Profession:**



A combo box list appears as a single row on your form. It allows the user to type in a choice or choose one from a popup list. Alternately, you can configure a combo box list to present a calendar to your users, with which they can pick a date.

- **Box lists**

**Profession:**

Doctor
Lawyer
Engineer
Accountant

A box list can be any size and displays choices in a scrolling list.

Use box lists if you:

- Want the user to select only one choice
- Want to save space on your form
- Want to have your application dynamically generate a list of choices and insert them into the form

- **Calendars**

**Date:**

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

A calendar displays a field for the user to type a date into and a calendar for the user to select a date. In either case, the date is displayed in the format of your choice.

---

## Creating check box lists

Use check boxes if you want the user to be able to select more than one of the choices you present. For example, you may want the user to 'check all that apply' or present a single choice that the user can activate or deactivate.

**Check all that apply:**

- Newspaper
- Radio
- Television

To create a check box list:

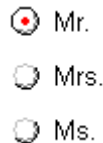
1. In the Palette, click **Check**.
2. Click on the canvas to insert the check box.
3. In the Palette, click **Label**.
4. Click on the canvas, beside the check box you added.
5. In the Properties view, expand **General**.
6. Click within the **value** field.
7. Type the label name.

- Repeat steps 1 through 7 to add more choices to the check box list.

---

## Creating radio lists

Radio buttons allow you to present a group of choices to the user from which only one choice can be selected at any time. For example, if you wanted to present the user with a choice that required them to choose their salutation, you could create three radio buttons, label them *Mr*, *Mrs* and *Ms* and put them in the same group. A user could select one or the other, but not both.



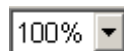
To create a radio button:

- In the Palette, click **Radio**.
- Click on the canvas to insert the radio button.
- In the Palette, click **Label**.
- Click beside the radio button on the canvas.
- Double-click the label and type the label for the radio button.
- Repeat steps 1 through 5 to add more choices to the radio button list.
- Select all the radio buttons.
- In the Properties view, expand **General**.
- Click within the **group** value field.
- Type the name for the radio button group. This name is applied to all selected radio buttons.

---

## Creating popup lists

Popup lists show only one choice at a time. The remaining choices are visible when the user clicks the arrow at the right of the list. The selected choice is always shown, but you can select a default choice to be visible before the user selects anything. You can also display text that indicates what the user is meant to choose from the list (for example, *Select a percentage*).



To create a popup list:

- In the Palette, click **Popup**.
- Click on the canvas to insert a popup list.
- In the Palette, click **Cell**.
- Click within the popup list on the canvas to add a choice to the popup list.
- In the Properties view, expand **General**.
- Click within the **label** value field.
- Type the label you want to display to the user. For example, *100%*.  
The **label** property shows the user either a single or multiple line text value.
- Click within the **value** field.

9. Type the value that will be stored in the form when the user selects the choice. For example, *100*.  
When the user selects the choice, the form will record the value, not the label name. The popup lists will also display the value when it is selected. The **value** does not apply to choices that trigger actions.
10. Repeat steps 3 through 9 to add other choices to the popup list.

---

## Creating combo box lists

Combo box lists act as a combination of a field and a list. Users can enter text, or choose from the list. Combo boxes can also have built-in labels. The following is an example of a combo box "expanded" to show the list of choices.



To create a combo box list:

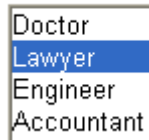
1. In the Palette, click **Combobox**.
2. Click on the canvas to insert the combo box list.
3. In the Properties view, expand **General**.
4. Click within the **label** value field.
5. Type the Ccombo box label to appear at the top of the combo box list. For example, *Profession*.
6. In the Palette, click **Cell**.
7. On the canvas, click within the combo box list to add a choice to the combobox list. The cell's **group** value is automatically set based on the name of the combo box. All cells in the same combo box have the same group value.
8. Click within the **label** value field.
9. Type the label you want to display to the user. For example, *Doctor*.  
The **label** property shows the user either a single or multiple line text value.
10. Click within the **value** field.
11. Type the value you want to store in the form when the user selects the choice. For example, *MD*.  
The **value** property refers to the contents of an item that will be stored in the form when the user selects the choice. If you set the value and the user selects the choice, then the form will record the value, not the label name. Also, in combo box lists, the choice will display the value when it is selected.
12. Repeat steps 6 through 11 add other choices to the popup list.

---

## Creating box lists

A box list is a box containing a list of choices. It differs from other lists in that you can adjust the size of the box list to display more than one choice at a time. The following is an example of a box list with a built-in label.

**Profession:**



Doctor
Lawyer
Engineer
Accountant

A box list can be as large as you want. If it contains more choices than can appear in the box at once, it automatically generates scroll bars.

To create a box list:

1. In the Palette, click **List**.
2. Click on the canvas to insert the box list.
3. In the Palette, click **Cell**.
4. On the canvas, click within the **box List** to add a choice to the box list. The cell's **group** value is automatically set based on the name of the box list. All cells in the same box list have the same group value.
5. In the Properties view, expand **General**.
6. Click within the **label** value field.
7. Type the label you want to display to the user. For example, *Doctor*.
8. Click within the **value** field.
9. Type the value you want to store in the form when the user selects the choice. For example, *MD*.
10. Repeat steps 3 through 9 to add other choices to your box list.

Tip: You can change the order of the choices in the list by opening the Outline view, selecting an item and dragging it up or down in the list.

---

## Creating calendars

Date picker calendars let the user type a date into a field, or use the calendar to select a date. In either case, the date is displayed in the format of your choice.

**Date:**



To create a calendar:

1. In the Palette, click **Date Picker**.
2. Click on the canvas to insert the calendar.
3. In the Properties view, expand **General**, **Format**, **format**, and **presentation**.
4. Set **calendar** to a calendar style. For detailed information on calendar styles, see the *Workplace Forms XFDL Specification* document.

---


## Using lists to trigger actions

Buttons are the most common way for users to trigger actions (for example, saving the form). You can also use cells within a popup list, combo box list, or box list to trigger actions.

Using lists to trigger actions is similar to using buttons to trigger actions. For detailed information on buttons, see "Buttons and actions" on page 67.

For a list of actions that can be triggered by buttons and cells, see "Buttons" on page 67.

To use a list to trigger actions:

1. Create a popup list, combo box list, or box list.
2. In the Outline view, select one of the list's cells.
3. In the Properties view, expand **General**.
4. Set **type** to an action type from the list. For detailed information on action types, see "Buttons" on page 67.
5. In the top-right corner of the Properties view, click  and click **Show Advanced Properties**.
6. In the Properties view, expand **Miscellaneous** and **Transmit**.
7. Set the properties for the action.

---

## Images

Use images on your forms to give them a unique look. You can use images to add department or corporate logos to the title, customize buttons or labels, or add small images like arrows to help users navigate through your form.

**Note:** You should consider the type, quantity, and size of the images you include on your forms as they can make a form larger and therefore slower to transmit, as well as have increased storage requirements. To help keep your images small, most graphics editing applications provide you with the ability to reduce file size by reducing the number of colors, reducing the resolution, altering the compression, changing to either black and white or gray scale.

The Designer supports the following image formats:

- Bitmap (.bmp)
- Joint Photographic Experts Group (.jpeg)
- Portable Network Graphics (.png)
- Graphics Interchange Format (.gif)
- Sun Raster (.rast)

**Note:**

- The Designer does not support transparency or animation and only supports the 87a format for .gif graphic formats.
- The Viewer supports fewer types of JPEG images than the Designer supports. A JPEG image that appears correctly in the Designer may not be displayed in the Viewer. If you plan to use JPEG images in your form, test one image in the Viewer to ensure that the JPEG type is supported.
- The Designer does not support progressive compression for JPEG images.

---

## Adding an image file to a form

Before you can add an image to a form or to an item (such as a button or a label), you need to enclose the image as a data file for a specific page in the Enclosures view.

To enclose an image file:

1. In the Enclosures view, select a page under **Data**. By default, every form has one page named **PAGE1**.
2. Right-click and select **Enclose File**.
3. Browse to the location of the image.
4. Select the file and click **Open**.

You can now add the image directly onto your form, or to label or button items on your form. When you copy the label or button using an image to another page in the form, the image is referenced to the original file.

**Note:** The information contained in **Pages** in the Enclosures view does not show you where the enclosure is displayed; it shows you where the actual data resides. Images are stored as data items. When you enclose a file, a data item is created for that specific page.

**Note:** You can also add an image file to a form by dragging the file from your Workspace or Windows Explorer onto the PAGE in the Enclosures view.

---

## Adding an image to a button or label

The Designer lets you customize your form by adding images to buttons and labels.

Before you can add an image to a button or label, you need to enclose the image file within the form. For information about enclosing a file, see “Adding an image file to a form” on page 81.

To add an image to a button or label:

1. Create a button or label.
2. In the Enclosures view, expand **Data** and the page that contains the image file (see “Adding an image file to a form” on page 81).
3. Drag the image file from the Enclosures view onto the button or label on the canvas. The image replaces any text on the button or label.

**Note:** You can create an image label by dragging an image file from the Enclosures view directly onto the canvas.


To resize or crop the image to better fit within the button or label, see “Resizing and cropping images on buttons and labels.”

To image map an image on a button, see “Image-mapping a button” on page 83.

## Resizing and cropping images on buttons and labels

You can resize or crop an image so it better fits within a button or label.

To resize or crop an image:

1. Select the button or label.
2. In the Properties view, click  and click **Show Advanced Properties**.
3. In the Properties view, expand **Appearance**.
4. Set **imagemode** to:
  - **clip** — If the image is smaller than the button/label, the image is centered on the button/label. Otherwise, the top left corner of the image is positioned in the top left corner of the button/label, and the parts of the image that extend beyond the border of the button/label are cut off.
  - **resize** — The image is resized to fit within and fill the button/label. The aspect ratio of the original image may be altered.
  - **scale** — The image is resized to fit within and fill the button/label. The aspect ratio of the original image is preserved.

**Note:** When you change the **imagemode** property for a label or a button, you are modifying how the image is displayed on the label or button, and not the actual image (data item).



## Image-mapping a button

When a user clicks a button that contains an image, the form automatically detects where the user clicked the image. You can use this feature to specify that a button performs a different action depending on where the user clicks the image.

This is an advanced feature. To use it, you must know how to hand-code formulas.

The form records the user's click based on an invisible grid that overlies the image. The points along each axis of the grid are numbered from zero through 1,000. The top left corner of the image is (0,0) and the bottom right corner is (1,000, 1,000), regardless of the size of the image. When the user clicks a button, the form determines the point on the grid that is closest to where the user clicks.

To image-map a button:

1. Create a button with an image on it (see "Adding an image to a button or label" on page 82 and "Creating buttons" on page 68).
2. Right-click the button and click **Wizards** → **Compute Wizard**.
3. From the **Property to Set** list, click **url**.
4. Click **The value is set by a manually created formula**.
5. Type the formula in the text area at the bottom of the window (see the examples below).
6. Click **Finish**.

### Examples

The following formula points to a different URL if the user clicks on the left half of the button or the right half of the button

```
coordinates[0]>"500" ? "http://www.myserver.com/Type1.xfd" :  
"http://www.myserver.com/Type2.xfd"
```

If the user clicks on the right half of the button (that is, the x-coordinate is greater than 500), the URL points to Type1.xfd. Otherwise, the URL points to Type2.xfd.

`coordinates[0]>"500"` indicates "if the x-coordinate is greater than 500".

The following formula points to a different URL if the user clicks on the top half of the button or the bottom half of the button

```
coordinates[1]>"500" ? "http://www.myserver.com/Type1.xfd" :  
"http://www.myserver.com/Type2.xfd"
```

If the user clicks on the bottom half of the button (that is, the y-coordinate is greater than 500), the URL points to Type1.xfd. Otherwise, the URL points to Type2.xfd.

`coordinates[1]>"500"` indicates "if the y-coordinate is greater than 500".

---

## Adding a background template image

If you have a paper form and are re-creating it in the Designer, you can scan the form and use the scanned image as a background or template in the Designer. The background image is only visible in the Designer; it is not visible in the Viewer and will not print.

**Note:** If you're replicating a paper form, specify which template image you will be using as a guide to assist you in laying out the form. Remember that the template image will stay the same size even if you use a different-sized window.

To add a background image to a form:

1. Scan a form and save it in one of the supported formats: JPEG, PNG, BMP, GIF, or Sun Raster.
2. In the Outline view, expand the page you want to add a background image to and select **Page Global**.
3. In the Properties view, expand **Appearance**.
4. Click within the **backgroundimage** value field.
5. Type the path and filename for the image (for example, c:\MyFile\MyImages\MyScannedForm1.bmp) and press Enter.
6. Click within the **backgroundimagealpha** value field.
7. Type a value between 1 and 255 to set the transparency of the background image. One (1) represents the lightest image background, and 255 the darkest.

---

## Formulas

Formulas enable your form to do mathematical or logical operations quickly. In addition to displaying calculations based on user input, formulas also allow your form to make decisions based on the results yielded by other formulas.

Formulas can send a user to a different page, determine an e-mail address or URL to which a form should be submitted, automatically calculate a compound interest factor, and so on.

---

### When to use formulas

Formulas are typically used in the following situations:

- When you need your form to act like a spreadsheet and perform mathematical operations on the values entered by the user.
- When you need the form to make a decision based on the values entered by the user. For instance, you may want the “dependents” section of a form to be made active or inactive based on whether the user has children.
- When you want to add dynamic elements to your form.

Common situations in which formulas are used include:

- Copying information from one field to another.
- Changing the text displayed by an item, or setting the value of a field.
- Changing whether an item is active or inactive.
- Changing the URL associated with a button or choice list (for transmitting the form).

For detailed information about formulas and items, see the *Workplace Forms XFDL Specification* document.

---

### Planning a formula

Before you create a formula in your form, review the following:

**Establish what you want the formula to do.**

Do you want the formula to perform basic mathematics, such as add, subtract, divide, or multiply values?

Do you want to use formulas to have your form make decisions based on user input?

**Establish where the data results will be written.**

Will the data be seen by the user in another field on the form?

Will the data be sent to a database, e-mail or URL?

**Establish the characteristics of the form, page and items.**

For example, the **bgcolor** property set at the form global level defines the background color of all pages on the form, whereas a *bgcolor* option set at the item level defines the background color for that specific item.

For detailed information about options, see “Functions” on page 91.

### Establish whether you will use a function.

For detailed information about functions, see “Operators and the order of operations” on page 96.

---

## Setting up simple formulas



The **Compute Wizard** allows you to perform the following basic mathematical operations:

- 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.

### Setting one value to equal another (assignment)

You can use an assignment formula to set any property. For instance, you can use this to set the value of a field or label equal to a value that the user typed. You can also use it to set a Submit button’s URL to an address that the user types in, or to activate certain items at the user’s request.

To set the value of an item equal to the value of another item:

1. Select the item for which you want to set up a formula. (The formula will set one of the item’s properties.)
2. Right-click the item and click **Wizards** → **Compute Wizard**.
3. From the **Property to Set** list, click **value**. For detailed information about other properties, see “Appendix B: Options” on page 187.
4. Click **The value is equal to another form item**.
5. Click **Next**.
6. Click . The canvas moves to the front for you to select an item.
7. Select the item you want to use as an input for the formula. The Compute Wizard window reopens.
8. In the field **below** the , click **value**.
9. Click **Finish**.

### Performing a calculation based on two values (calculation)

You can use these formulas to calculate the numerical or text values of items on a form.

#### Performing a calculation based on two numbers or text values





To perform calculations based on two numbers or text values:

1. Right-click the item and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, select the property of the item that you want to set using a formula. For detailed information about properties, see “Appendix B: Options” on page 187.
3. Click **The value is equal to the calculations of two values**.
4. Click **Next**.

5. From the **First Value** list, click **Enter a number or text**.
6. Type the number or text you want to use.
7. In the **Function** list, select the operator you want to use. For detailed information about operators, see “Operators and the order of operations” on page 96.
8. From the **Second value** list, click **Enter a number or text**.
9. Type the number or text you want to use.
10. Click **Finish**.

## Performing a calculation based on two items on the form

To perform calculations based on two items on the form:

1. Right-click the item and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, select the property of the item that you want to set using a formula. For detailed information about properties, see “Appendix B: Options” on page 187.
3. Click **The value is set by a calculation of two values**.
4. Click **Next**.
5. From the **First Value** list, click **Choose an item on the form**.
6. Click . The canvas moves to the front for you to choose an item.
7. Within the field right of , select a property. For detailed information about properties, see “Appendix B: Options” on page 187.
8. Select the item you want to use as an input for the formula. The Compute Wizard window reopens.
9. In the **Function** list, select the function you want to use. For detailed information about functions, see “Functions” on page 91.
10. From the **Second Value** list, click **Choose an item on the form**.
11. Click . The canvas moves to the front for you to choose an item.
12. Select the item you want to use. The Compute Wizard window reopens.
13. Within the field right of , select a property. For detailed information about properties, see “Appendix B: Options” on page 187.
14. Click **Finish**.

## Performing a calculation based on an advanced calculation

To perform an advanced calculation:

1. Right-click the item and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, select the property of the item that you want to set using a formula. For detailed information about properties, see “Appendix B: Options” on page 187.
3. Click **The value is equal to the calculation of two values**.
4. Click **Next**.
5. From the **First Value** list, click **Choose a function**. The Function window is displayed.
6. Click **Function**.


7. From the **Choose a Function** list, select a function. Depending on what function you choose, different **Parameters**, **Descriptions**, and **Return Value** information are displayed. For detailed information about functions, see “Functions” on page 91.
8. When presented with **Parameter** options, select the appropriate settings.
9. Click **OK**. The Compute Window is displayed.
10. From the **Second Value** list, click **Choose a function**. The Function window reopens.
11. From the **Choose a Function** list, select a function. Depending on what function you choose, different **Parameters**, **Descriptions**, and **Return Value** information are displayed. For detailed information about functions, see “Functions” on page 91.
12. Click **Finish**.

For detailed information on referencing paths, see “References: Referring to other items and their options” on page 97.

## Summing values

You can use a formula to add the values of a number of different fields on the form and display the result in another field. This formula type is only useful for setting field contents to display the sum of the contents of other fields.

To add multiple fields:


1. Right-click the field that will contain the total and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, click **value**.
3. Click **The value is equal to the sum of multiple fields on the form**.
4. Click **Next**.
5. Click . The canvas moves to the front so you can choose the items to use.
6. Hold down Ctrl and select the items you want to add together.
7. Once your selection is made, click **Finish**. To delete items from your selection, highlight the item in the **Compute Wizard** list and click **Delete**.
8. In the **Compute Wizard**, choose the appropriate operator. For detailed information on operators, see “Operators and the order of operations” on page 96.
9. Click **Finish**.

## Setting an item value to equal a function

Functions are complex, preset formulas that make your form more powerful and flexible. Functions allow forms to perform procedural logic, and also to perform operations that would normally require complex conditional statements to achieve.

To set an item value to equal a function:

1. Right-click the item and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, click **value**.
3. Click **The value is equal to a function**.
4. Click **Next**.
5. Click **Function** to open the Function Call window.

6. From the **Choose a Function** list, select a function. Depending on what function you choose, different **Parameters**, **Description**, and **Return Value** information displays. For detailed information about functions, see “Functions” on page 91.
7. Review the **Description** to confirm that you have selected the appropriate function.
8. Under **Parameters**, you can do either of the following:
  - **Enter a number or text:** type the value into the parameter field
  - **Choose an item on the form:** click . The canvas moves to the front for you to choose the item to use. Once chosen, select a parameter option and click **OK**.
9. Click **Finish**.

## Making decisions based on user input (if/then/else)

An if/then/else statement lets you set the property of an item based on user input. For example, you can set up a field to display different information based on whether a check box option is selected, or to set a button to be active if a radio button option is selected.



To create an if/then/else statement:



1. Select the item that will contain the formula.
2. Right-click the item and click **Wizards** → **Compute Wizard**.
3. From the **Property to Set** list, select the property of the item that you want to set using a formula. For detailed information about properties, see “Appendix B: Options” on page 187.
4. Click **The value is determined by a decision (If/Then/Else)**.
5. Click **Next**.

The Formula window is now configured to set up an if/then/else formula. You can compare only combinations of numerical values, text, properties of items on the form, and the results of function calls.

If you are setting up the formula to determine user input, set the first element of the formula to the property of an item on the form.

To select an item on your form:

1. From the **If** list, click **Choose an item on the form**.
2. Click .
3. Click the item that contains the property you want to use.
4. In the menu over the first field in the **If** statement, select the type of property you would like the formula to check. For detailed information about properties, see “Appendix B: Options” on page 187.
5. Decide if you want the formula to check against a numerical value, text, the result of a function or a property of another item.
  - If you wish to check against a numerical value or text, click **Enter number or text** from the menu above the last field in the **If** statement.
  - If you wish to check against the result of a function, click **Choose a function** from the menu, then in the Function window, choose the function you want and type in the parameters.
  - If you wish to check against the property of another item, click **Choose an item on the form** and click  to choose the item from the form.

- If you wish to check if the item is set, click **on** or **off**. For example, you may want to apply this *if* statement to a **Radio** or **Check** item.
6. From the **Then** list, select what to assign the property if the *If statement is true*.
    - If you wish to assign a numerical value or text, click a number or text from the menu.
    - If you wish to assign the result of a function, click a function from the menu, then in the Function window, choose the function you want and type in the parameters.
    - If you wish to assign a property of another item, click **Choose an item on the form** and click  to choose the item from the form.
  7. In the menu above the field in the **Then** statement, choose what to assign the property if the *If statement is false*.
    - If you wish to assign a numerical value or text, click a number or text from the menu.
    - If you wish to assign the result of a function, click a function from the menu, then in the Function window, choose the function you want and type in the parameters.
    - If you wish to assign a property of another item, click **Choose an item on the form** and click  to choose the item from the form.
    - If nothing happens after meeting the *If* argument, leave this field empty.
  8. Click **Finish**.

For detailed information about logical operators, see “If/then/else logical operators.”

Review a sample of an If/Then statement — “Example of an if/then/else formula” on page 91.

### If/then/else logical operators

Logical formula operators allow you to compare values and evaluate the results. When two values are compared using logical operators, the result is either true or false.

The following logical operators are available in the Compute Wizard if/then/else formula menu:

Logical operator	Description
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	equal to (comparison)
!=	not equal to
&&	AND
and	AND
	OR
or	OR
!	NOT



When two values are compared using logical operators, the result is either true or false. For instance, if you have a form with a check box, you can use the following logical statement to see if the check box has been selected:

```
CHECK1.value == 'on'
```


You can then create a formula that will instruct the form to perform one action if the logical statement evaluates to true, and another action if the logical statement evaluates to false, such as this, which sets the text of a field:

```
CHECK1.value== 'on' ? ('Check box is on') : 'Check box is off'
```

You use logical operators in if/then/else formulas. The Designer can set up simple if/then/else formulas, in which one logical statement, containing only one logical operator, is evaluated. For detailed information about making decisions based on user input (if/then/else), see “Making decisions based on user input (if/then/else)” on page 89.

### Example of an if/then/else formula

To create a compute that makes FIELD1 visible if CHECK1 is selected by a user:

1. Right-click FIELD1 and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, click **visible**.
3. Click **The value is determined by a decision**.
4. Click **Next**.
5. From the **If** list, click **Choose an item on the form**.
6. Click  and click CHECK1.
7. When you return to the Compute Wizard window, click **value**.
8. From the operators list, select **== (equal to)**.
9. From the list to the right of the options list, click **on**.
10. From the **Then** list, click **on**.
11. From the **Else** list, click **off**.
12. Click **Finish**.

---

## Functions

Functions are complex, preset formulas that make your form more powerful and flexible. Functions allow forms to perform procedural logic, and also to perform operations that would normally require complex conditional statements to achieve.

Functions are compiled into packages (libraries), that must reside on the user's computer.

For detailed information about standard mathematical operations, string manipulations, and logical operators used in formulas, see the *XFDL Specification* and the *Introduction to the Viewer Functions*.

The following functions are available in the **Compute Wizard**.

**abs** Returns the absolute value of a number.

**acos** Returns the arc cosine of a number in radians.

**annuity**

Returns the annuity factor for a specified rate and number of periods.

**applicationName**  
Returns the name of the currently running application. Does not require parameters.

**applicationVersion**  
Returns the version of the currently running application in a string format. Does not require parameters.

**applicationVersionNum**  
Returns the version of the currently running application in numeric form. Does not require parameters.

**asin** Returns the arc sine of a number in radians.

**atan** Returns the arc tangent of a number in radians.

**ceiling**  
Returns the integer ceiling (the next highest whole number) of a number.

**checkValidFormats**  
Returns the number of items in the form that have invalid formats.

**compound**  
Returns the interest factor for a specified rate and number of periods.

**cos** Returns the cosine of a number in radians.

**countChildren**  
Returns the number of children for the current node.

**countDatagroupItems**  
Returns the number of items in a particular data group. The parameter requires the name of the grouped item. The parameter requires the name of the grouped item. Selecting non-grouped items will create a null entry.

**countGroupedItems**  
Returns the number of items in a particular group. The parameter requires the name of the grouped item. Selecting non-grouped items will create a null entry.

**countLines**  
Returns the number of lines that the string would occupy for a given width.

**countWords**  
Returns the number of words in the supplied string.

**date** Returns the current date in yyymmdd format. Does not require parameters.

**dateToSeconds**  
Returns the number of seconds since 00:00:00 GMT, Jan. 1, 1970 for a specified date and time.

**day** Returns the numeric day of the month. Does not require parameters.

**dayOfWeek**  
Returns the numeric date of the week (Sunday is 1). Does not require parameters.

**decimal**  
Returns the decimal (base 10) version of a specified number in a specified base.

**deg2rad**  
Returns the number of radians in a number of degrees.

**destroy**  
Destroys the specified form element.

**duplicate**  
Duplicates the specified form element.

**endOfMonth**  
Returns the number of seconds of an end-of-month day from 00:00:00 GMT, Jan. 1, 1970.

**exp** Returns the exponentiation of a number.

**fact** Returns the factorial of a given integer.

**floor** Returns the integer floor (the next lowest whole number) of a number.

**for** Counts through a series of numbers like a traditional for loop in programming.

**forLoop**  
Creates a loop that you can use to run a compute a number of times.

**formatString**  
Returns a specified string with a specified format applied (for example, dollar format).

**generateUniqueName**  
Returns a unique name that can be assigned to a new form element, such as a duplicated element.

**get** Returns the value of a specified form option.

**getAttr**  
Returns the value of an attribute on a form element.

**getDateByPath**  
Returns data associated with a DSOM object.

**getGroupedItem**  
Returns the SID of an item in a particular group.

**getInstanceRef**  
Returns the reference to a particular data instance in the form's XML data model.

**getPosition**  
Returns a number that represents which child the item is. This count is zero-based. For example, the first child of an item would return a 0, the second child a 1, and so on.

**getPref**  
Retrieves the preferences entry for the **Viewer**.

**getReference**  
Returns the reference to the form element containing the function.

**hour** Returns the current hour in hh 24-hour format.

**isValidFormat**  
Returns the result of comparing a string against an item format.

**ln** Returns the natural log of a number.

**log** Returns the log of a number to a specified base.

**minute** Returns the current minute in mm format. Does not require parameters.

**mod** Returns the modulus of a number with the specified divisor.

**month** Returns the current month in numeric format.

**now** Returns the number of seconds since 00:00:00 GMT, Jan. 1, 1970. Does not require parameters.

**pad** Returns a string that is either padded (spaces added) or truncated.

**pi** Returns pi to six decimal places. Does not require parameters.

**power** Returns the number raised to the specified power.

**rad2deg** Returns the number of degrees in a number of radians.

**rand** Returns a random integer between two bounds inclusively.

**replace** Returns a string with a portion replaced by another string. The string is treated as an array; the start and end points are identified by array element numbers.

**round** Returns a decimal number rounded to a specified decimal place.

**second** Returns the current second in ss format. Does not require parameters.

**set** Sets the value of an option in the form.

**setAttr** Sets the attribute to "1" if the operation completed successfully or "0" if an error occurred.

**sin** Returns the sine of a number of radians.

**sqrt** Returns the square root of a given number.

**strlen** Returns the length of a string.

**strmatch** Determines if a wildcard string matches a regular string.

**strpbrk** Returns the first position of any character from one string in another string.

**strrstr** Returns the position of the last instance of one string in another string.

**strstr** Returns the position of the first instance of one string in another string.

**substr** Returns a substring of a string. The string is treated as an array; the start and end points of the substring are specified by numbers corresponding to array elements.

**tan** Returns the tangent of a number where the number is expressed in radians.

**time** Returns the time specified (or current time if no time specified) in hh:mm AM format.

**toggle** Detects transitions in a property. Toggle is frequently used to set the value of a property based on radio button or check box selection, button click or field entries.

**toLowerCase**  
Returns the lower case form of a string.

**toUpperCase**  
Returns the upper case form of a string.

**trim** Returns a string with all leading and trailing white space removed.

**URLDecode**  
Returns the URL-Decoded form of a string. (For example, takes the % out of URLs and replaces them with spaces.)

**URLEncode**  
Returns the URL-Encoded form of a string. (For example, replaces spaces with the % character.)

**xmlModelUpdate**  
Updates the XML model in the form.

**xmlModelValidate**  
Validates the XML model against schema referred to in the form or provided as part of the form.

**year** Returns the current year in yyyy format. Does not require parameters.

**viewer.addressBook**  
Uses MAPI to bring up your mail client's address book.

**viewer.env**  
Returns a string that contains the details of the environment in which the Viewer is operating.

**viewer.fileOpen**  
Returns the name of a file selected for saving through a file browser window.

**viewer.fileSave**  
Returns the name of a file selected for opening through a file browser window.

**viewer.getDefaultFilename**  
Returns the default name of the currently activated form.

**viewer.getHeight**  
Returns the current height of the item in pixels or characters.

**viewer.getHelpMode**  
Returns the current help mode setting of the Viewer.

**viewer.getWidth**  
Returns the current width of the item in pixels or characters.

**viewer.getX**  
Returns the current location of the item's left edge in pixels.

**viewer.getY**  
Returns the current location of the item's top in pixels.

**viewer.measureHeight**  
Calculates how tall an item would have to be to display all of its text.

**viewer.messageBox**  
Displays a simple message box with a caption and message.

**viewer.param**  
Returns the value of the HTML param element's value attribute.

**viewer.setCursor**

Sets the cursor to the given index and selects the text if a range is provided. The cursor is set to the end of the field if no parameters are provided.

**viewer.setDefaultFilename**

Sets Save/Mail default name of the activated form.

**viewer.setHelpMode**

Sets or clears viewer help mode.

**viewer.showCalendar**

Open a calendar widget that allows the user to select a date. This widget appears attached to the item that contains the function call.

## Operators and the order of operations

The following operators are available when setting a formula.

Operator	Description
+	addition (or concatenation, if text)
-	subtraction (or negative, if placed in front of an integer)
*	multiplication
/	division
^	exponentiation
%	percentage
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	equal to (comparison)
!=	not equal to
&&	AND
and	AND
	OR
or	OR
!	NOT
x?y:z	if x then y, else z
=	assignment
.	structure membership
[]	array membership
->	indirect membership

### Order of Operations

Operations are evaluated in the following order:

1. Membership
2. Exponentiation
3. Multiplication, division, and unary minus

4. Addition and subtraction
5. Relational (greater than, less than, equal to, and so on)
6. Logical AND
7. Logical OR
8. Logical IF THEN ELSE

---

## Creating custom formulas

There are an endless number of possibilities when using formulas. You can create formulas that make decisions, add numbers, concatenate strings, or simply copy a value from one field to another. The best way to master the possibilities is to experiment. You can also use a custom formula to access a user's choice list using the indirect item reference.

Sometimes, you will need to set up formulas that contain more than one logical operator, called a custom formula. When you use more than one logical operator in a custom formula, you need to let the form know that the logical statement being evaluated has more than one element, and the result, which the form uses to determine the next action, comes from the combined statement.

For example, if you had two check boxes that need to be selected for an action to occur, then the formula to set the action's active property to "on" would be as follows:

```
(CHECK1.value == 'on' and CHECK2.value == 'on') ? ('on') : 'off'
```

This ensures that the form evaluates the entire statement before deciding what to do.

If you want to enter formulas directly into the source code or Properties view, you will need to refer to the *XFDL Specification*.

For detailed information about how to write formula references manually, see "References: Referring to other items and their options" and "Custom functions" on page 101.

## References: Referring to other items and their options

For advanced users: If you write your own formula you need to be familiar with how to reference other items on your form.

References allow you to identify a specific option by providing a "path" to it. This means that you can refer to an option anywhere in the form.

There are two types of references:

### **direct reference**

Refer directly to a specific option anywhere in the form.

For example, to refer to the value of the field `total_field` on page 1 of the form, you would use a direct reference: `page1.total_field.value`

### **indirect**

Reference a choice made in a list.

For example, to refer to the selected choice in a popup list called `dept_popup`, you would use an indirect reference: `dept_popup.value->value`

A reference is constructed by combining the page tag, item tag, and option name that references to the option you want.

**page (Palette item)**

The name, for example the **sid**, you have given to the page. This is only necessary if the item you are referring to is on a different page.

**item (Palette item)**

The name, for example the **sid**, you have given to the item. This is only necessary if the property you are referring to is in a different item.

**[n] or [name] (Option names)**

The position in the array of the array element, if you are referring to a property that is defined by an array. This is either a number or a variable name.

-> The dereference symbol (indirect membership). This is used when the reference is to a list choice.

**Sample References**

The following samples illustrate how a reference might actually look. In each case, assume the names in quotations are either page tags or item tags.

`page_1.submit_button.value` Indicates the value option for the “submit\_button” on “page\_1”.

`age_field.itemlocation[2]` Indicates the second itemlocation (the “y” coordinate) for the “age\_field” on the current page.

`popup_1.value->value` Indicates the value of the list choice of “popup\_1”.

Tip: To avoid confusion, always use the full path, including Page and item Sid.

---

## Deleting a formula

To delete a formula:

1. Select the item that has the formula applied to it.
2. In the Properties view, expand **Appearance** and **visible**.
3. Click within the **compute** value field to highlight the contents.
4. Press Delete.

---

## Formula examples

The following examples will help you in creating your own custom formula.

### Automatically calculating compound interest factor

Learn how to create a form where the user, using the Viewer, can enter a dollar figure for the starting amount of the loan in the first field, enter the period in the second field, see the compound interest factor in the third field, and get the total amount of the loan in the fourth field.



### Workflow overview:

1. Set up items onto the form.
  - Complete Steps One - Three
2. Use **Compute Wizards** to set the formulas.
  - Complete Steps Four - Five

### Step One: Create a new form

First, make sure that you have a blank form open in the Designer.

1. Click **File** → **New** → **New Workplace Form**.

For detailed information about creating forms, see “Creating forms” on page 17.

### Step Two: Add items to the form


1. From the Palette, click **Field** and then click on the canvas to add this field to the form.
2. Add three more fields below the first one.

### Step Three: Set the properties for the first and second fields

1. Select the first field.
2. In the Properties view, expand **General**.
3. Click within the **label** value field
4. Type: *Original Amount of Loan* and press Enter.
5. In the Properties view, expand **Format** and **datatype**.
6. Set the **datatype** to **Currency**.
7. Select the second field and repeat steps 2 through 5 to label the second field *Period* and set its **datatype** to *float*.

You have now set up the first two fields so the user can enter the starting amount and the length of time for the loan in question.



### Step Four: Set the third field to calculate the interest factor

1. Right-click on the third field and click **Wizard** → **Compute Wizard**.
2. From the **Property to Set** list, click **value**.
3. Click **The value is equal to a function** and click **Next**.
4. Click the **Function** button. The Function window opens.
5. From the **Function Call** list, click **compound**.
6. In the **Parameters rate** list, click **Enter a number or text**.
7. Tab to the **Enter the Value** field and type the interest rate *.07*.
8. In the **Parameters period** list, click **Choose an item on the form**.
9. Click  to return to the canvas, and click on the field labeled *Period*.
10. Click **OK** to exit the Function Call window.
11. Click **Finish** to apply the formula to the field.

The third field will now calculate the compound interest factor based on a rate of 7% and a period of whatever number is entered into FIELD2.

### Step Five: Set the fourth field to calculate the total value of the loan

Set up the fourth field to show the total value of the loan by multiplying the starting amount by the compound interest factor.

1. Right-click on the fourth field and click **Wizard** → **Compute Wizard**.
2. From the **Property to Set** list, click **value**.
3. Click **The value is set by a calculation of two values** and click **Next**.
4. From the **First Value** list, click **Choose an item on the form**.
5. Click  to go back to the form, and click on the first field.
6. An option list is displayed. Click **value** from the option list.
7. From the **Function** list, select **\*** (**multiplied by**).
8. From the **Second Value** list, click **Choose an item on the form**.
9. Click  to go back to the form, and click on the third field.
10. An option list is displayed. Click **value** from the option list.
11. Click **Finish** to exit the Formula window.

## Displaying the current date automatically

You can set up a field to automatically display the current date when the form opens.

To display the current date automatically:

1. Select the item.
2. In the Properties view, expand **Format**, and **format**.
3. Set **datatype** to **date**.
4. Right-click the item and click **Wizards** → **Compute Wizard**.
5. From the **Property to Set** list, click **value**.
6. Click **The value is equal to a function**.
7. Click **Next**.
8. Click **Function**. The Function window is displayed.
9. Set the **Function call** to **date**.
10. Click **OK**.
11. Click **Finish**.

---

## Custom functions

Functions are complex, preset formulas that make your form more powerful and flexible. They allow forms to perform procedural logic, and also to perform operations that would normally require complex conditional statements to achieve.

The Designer includes two libraries of predefined functions called *system* and *viewer* functions. In addition, the Designer allows you to use your own custom functions. These functions can be executed when running the form in the Viewer, on Webform Server or in API mode. For detailed information about functions included with the Designer, see “Functions” on page 91.

---

## Creating custom functions

Before trying to set up your own custom function, learn about references, formula operators, the order of operations, and logical operators included with the Designer.

If a custom function is required, a forms developer can create custom function libraries to add additional functionality to a form. These custom functions can be written in C or Java.

If the custom functions are written in Java, these functions must be saved in a specialized compressed Java Archived file format (\*.jar). The following contents are enclosed in a JAR file:

- Manifest file — Used, during run-time, by the Viewer to locate the extension file.
- Extension file — Is the wrapper file that lets the Viewer access your custom functions.
- Function definition file — Defines your functions.

JAR files can be either embedded into the form or distributed on all client machines.

If the custom functions are written in C, these functions are saved in the Interactive Financial eXchange file format (\*.ifx) and installed on all client computers using the form. IFX files are usually TIFF files and can often be opened in an image editor.

For detailed information on how to develop your own custom functions, see the *Workplace Forms Java API User's Manual*.

---

## Making custom functions available

Functions are compiled into packages (libraries). Custom functions can be used by the form by using either of these methods:

- Call can be made to an install module on an individual computer(s) - **IFX** or **JAR** file.
- Call can be made to an embed module in the form - **JAR** file.
- Call can be made to a Web service.

**Note:** Web services lets one application talk to another server using the Internet. For example, using Web services you can create functions that let the Viewer communicate with server-side applications and update information in a form without the user having to submit the form. For detailed information on Web services, see “Web services” on page 127.

## Distributing IFX files

The Designer lets you reference an IFX file containing custom functions written in C.

Install the \*.ifx file on the client computer(s) in the Workplace Forms Viewer *extensions* folder.

For example, C:\Program Files\IBM\Workplace Forms\Viewer\2.6\extensions\

Referring to these modules allows you to use custom functions in a form without increasing the file size of the form itself. However, this requires you to distribute the appropriate IFX file to all of your users. Using this method of distribution has less security issues.

## Embedding JAR file

The Designer lets you reference a JAR that is installed on a user’s machine or a JAR that is embedded in a form.

**Note:** JAR files can be installed in the same extensions folder as IFX files.

Embedded custom function Java modules will increase the file size of your form and add higher security checks.

To embed a custom function module:

1. From the Enclosures view, expand **JAR**.
2. Right-click on one of the pages in the form, for example **PAGE1**, and select **Enclose file**.
3. Browse to find the \*.jar file.
4. Click **Open**. The custom functions are available throughout the entire form.

---

## Using custom functions in the Compute Wizard

In order to use the **Compute Wizard** to set up a call to a custom function, you need to know the package name, function name and a list of parameters. For example:

```
sample_package.convertDate(theDate.value, theLocale.value)
```

where sample\_package is the package name, convertDate is the function name and within the brackets is a list of parameters.

To set up a call to a custom function:

1. Right-click on the item you want to apply a custom function to and click **Wizards** → **Compute Wizard**.
2. From the **Property to Set** list, select a value. For detailed information about properties, see “Appendix B: Options” on page 187.

3. Click **The value is set by a manually created formula**.
4. Enter the formula containing the custom function call in the file provided.
5. Click **Finish**.

**Note:** To complete the above steps, the IFX file or JAR file does not need to be installed on your machine or embedded within the form. The functions do not need to be available to the form while the form is being designed.



---

## Attachments

An *attachment* is a separate file that is attached to a form. An attachment could be a Microsoft® Word document, an HTML document, an image, or any other type of document or file. Attachments are useful in the following situations:

- Attaching a document to a form so users can easily access and view the document. For example, an insurance application form may have an attached brochure containing detailed information describing the available insurance plans.
- Allowing users to attach documents to a form. For example, an employment application form may require the user to attach their resume.

Attachments are not displayed on the form. Attachments are stored within the actual form, unlike e-mail attachments which are separate files. After you attach a file to a form, changes to the original file on the user's computer will not affect the file attached to the form.



Attachments are stored in file folders within the form. Before you design a form, plan the folders you intend to use. For example, if you are creating an employee form, you may want to create two folders: one for performance evaluations and one for the employee's history. You may also want to limit which folders users can access. During form design, you define these folders as you attach files and setup attachment buttons.

Attachment buttons let users attach files to a form, save attachments to their computer, display attachments, or remove attachments from a form.

---

### Attaching files to a form

To attach a file to a form:

1. In the Enclosures view, expand **Data**.
2. Select the page that will contain the attachment.
3. Right-click the page and click **Enclose File**.
4. Use the browser to select the file to attach and click **Open**. The attached file is listed in the Enclosures view under the page.
5. Select the attached file.
6. In the Properties view, expand **General** and **datagroup**.
7. Click  within the **Data Group Refs** value field to add a **datagroupref** property.
8. In the **datagroupref** field, type the name of the folder that will contain the attachment.
9. If you want the attached file to belong to more than one folder, click within the **Data Group Refs** field and click  to add additional **datagroupref** properties.

---

## Creating attachment buttons

Attachment buttons let the user:

- attach a file to a form
- display an attachment (a file that is attached to a form)
- extract an attachment from a form and save it as a file
- remove an attachment from a folder or from the form.

**Note:** You can use cells, instead of buttons, to let users work with attachments. The procedure for creating attachment cells is similar to the procedure for creating attachment buttons. For detailed information on cells, see “Lists and choices” on page 75.

**Note:** If your form uses XForms, you can also use the XForms **Upload** item to let users attach a file to a form. For detailed information on XForms, see “XForms” on page 129.

To create an attach, display, extract, or remove button:

1. Create a button and select it. For detailed information on creating buttons, see “Creating buttons” on page 68.
2. In the Properties view, expand **General**.
3. Click in the **type** value field and click one of the following:
  - **enclose** — attaches a file to a form
  - **display** — displays an attachment (in an application determined by the attachment’s MIME type)
  - **extract** — extracts an attachment and saves it as a file
  - **remove** — removes an attachment from a folder (if the attachment belongs to more than one folder) or from the form (if the attachment belongs to only one folder).
4. Expand the **datagroup** property.
5. Click within the **Data Group Refs** value field.
6. Click **+** to add a **datagrouppref** property.
7. In the **datagrouppref** value field, type the name of the folder that you want the action to access. Folder names can include uppercase letters, lowercase letters, numbers, and underscores.
8. To allow the action to access additional folders, click within the **Data Group Refs** value field and click **+** to add additional **datagrouppref** properties. Otherwise, to setup the button to display, extract or remove a specific file that is already attached to the form, set **data** to the name of the attachment.

You can use an enclose button in the Preview to attach files to a form; however, the attachments will not be saved with the form when you return to Design view. To save the attachments with the form, in the Preview, click the **Save Form** button.



---

## Signatures

Electronic signatures essentially “lock” the data to the form, providing the following services:

- An electronic signature provides security. This functionality is built into the technology of the signatures themselves, which causes the signature to break if the document is changed after it is signed.
- Similar to a handwritten signature, an electronic signature indicates agreement with the document that is signed. When you sign a document, you are agreeing to the contents of the document. For example, when you sign a withdrawal slip at a bank, you are agreeing to withdraw a certain amount of money, and when you sign an employment contract, you are agreeing to abide by the rules established by your employer in that document.
- An electronic signature also identifies the signer. This identification occurs in a number of ways, depending on the type of signature you use. However, in all cases they provide a mechanism for tracing the signature back to the signer.

**Note:** While all electronic signatures satisfy these requirements in some way, some do a better job than others. For detailed information about types of signature engines, see “Signature types” on page 109.

Electronic signatures do not prevent people from tampering with a document; they simply make it obvious when tampering occurs. Tampering with the signed data causes the signature to break, which lets you know that you cannot trust the document.

Signatures do this by storing a hash of the form when it is signed. You can think of this hash as a snapshot of the form, showing exactly what the form looked like when it was signed. The next time the form is opened, it compares its stored snapshot to a new snapshot of the form, and determines if there are any changes. If there are changes, the signature will break, making it obvious to the user that some changes were detected and the form should not be trusted.

In an effort to reflect the intent of a signature, Workplace Forms Viewer actively stops people from making changes to signed data. However, there is no way to stop somebody from opening the form in a text editor and making changes to signed data. In this case, although the change cannot be prevented, the signature will break, alerting you to the change.

In addition, some of the forms you create may require signatures by more than one person, contain several sections for different people to fill out and sign, or some elements might be required to be excluded from signing altogether. You can use signature filtering to configure forms for layered and incremental signing, and also for the exclusion of elements (see “Creating a signature button that signs part of a form” on page 122).

For more information on signatures, see the *Workplace Forms Creating Signature Buttons in XFDL* document.

**Note:** In all cases, IBM strongly advises that you consult legal counsel to help determine your particular requirements with respect to the use and implementation of electronic signatures.

## Electronic signatures versus encryption

Electronic signatures essentially lock the data on a form so that it is obvious when tampering occurs. Tampering with the signed data causes the signature to break, which lets you know that you should not trust the document. Electronic signatures do not encrypt the data on a form in any way; they do not prevent people from reading the information. In fact, this would defeat the purpose of the signature, which is to indicate agreement with the information provided.

**Note:** If encryption is a concern, you must take other measures, such as implementing SSL security on your web site, or creating a Viewer extension that encrypts the form before it is submitted.

---

## Signing a form

A user can sign a form by clicking a signature button on the form. You usually design a signature button with some text or label that indicates its function (for example, "Click to Sign"). Typically, signature buttons are also larger than other buttons because they have to display the signer's name after the form is signed.

When a user clicks a signature button, the Digital Signature Viewer opens. The Digital Signature Viewer lets users sign forms, verify or delete existing signatures, and view the details of what parts of the form were signed. Users click "Sign" to sign the form. If they have more than one digital certificate installed on their computer, a dialog box appears, displaying all of the available signatures from which they can choose only one.

After the user selects a certificate, the form is signed and the Digital Signature Viewer displays the details of the signature. The user can then return to the form. In the form, the signature button changes to reflect the new signature. Typically, it displays the name and e-mail address of the signer (although this depends on the type of signature engine used). Once a form is signed, the Viewer prevents users from changing any of the signed information. Furthermore, when the user mouses over a signed item on a form, a tooltip indicates that the item is signed and cannot be altered.

**Notes:** When a user clicks a signature button, the Viewer automatically creates the following elements on the form:

- A **signer** option is added to the signature button. In general, this option contains the signer's name and e-mail address. However, the value assigned to this option depends on the signature engine that was used to sign the form.
- A **signature** item is created on the same page as the signature button. This item is given the name indicated in the button's signature option, and is used to store the details of the signature. You do not need to create the signature item yourself. The signature item will also include any of the filtering options used in the signature button.

A form may require any number of signatures. For example, a purchase order form may have two sections, one for the customer and one for an internal staff member who processes the order. The customer might fill out the first half of the form, and then add a signature that covers that portion of the form. The internal staff member might then review the form, and add a second signature that covers the entire form.

For detailed information about signing forms, see the *Workplace Forms Viewer User's Manual* document.

---

## Signature types

Electronic signature refers to any signature that is created electronically, and it describes a category of technology. Within that category, there are many types of electronic signatures. The Designer supports a number of different signature types (signature engines). When you create a signature button, you must configure it to use a specific signature engine.

### Digital signatures

Digital signatures are among the most secure electronic signatures. When you use digital signatures, each user is given a digital certificate. This certificate is actually a small file on a disk or on another device, such as a smart card. Each certificate also contains a unique code, and the certificate imprints this code on each signature you create with it. This means that all of your signatures can be traced back to your certificate, and the certificate itself can be traced back to you. In this way, digital signatures identify you through a clear chain of ownership.

Workplace Forms supports the following digital signature types:

- **RSA standard signatures** — These signatures are based on the RSA standard for digital signatures. This is a public standard that is broadly supported by both Public Key Infrastructure (PKI) and browser vendors. Workplace Forms products rely on the security libraries in the Microsoft Internet Explorer and Netscape browsers to provide support for RSA signatures.
- **Entrust signatures** — These signatures are based on a proprietary standard developed by Entrust, Inc. These signatures are not broadly supported, and require additional software from Entrust. This may take the form of client software or a central server that processes online signature requests.

In general, decisions about whether to use RSA signatures or Entrust signatures relies on individual preferences about their comparative strengths and features.

### When to use digital signatures

Digital signatures are best used for controlled groups who require tight security. Remember that each user must receive a certificate, and further that each user must keep that certificate safe from theft and copying. This means that applications for the general public are not normally good candidates for digital signatures unless there is a large body already distributing certificates to the public (such as the government).

Digital signatures also incur significant overhead. To issue and track digital signatures, you need a Public Key Infrastructure (PKI), which can be costly and time consuming to maintain. In general, most organizations will not adopt this sort of system unless they have a strong security need (or are mandated through law). For example, military organizations or other government agencies might use digital signatures.

### Generic RSA signatures

Generic RSA signatures are a type of digital signature (see “Digital signatures”). The Generic RSA engine uses a standard encryption algorithm that supports both

the Microsoft and Netscape signature engines. The Generic RSA signature uses digital certificates from either your Microsoft Internet Explorer or your Netscape certificate store.

## Entrust signatures

Entrust signatures are a type of digital signature (see “Digital signatures” on page 109). Entrust signatures let the user sign the form using Entrust certificates.

**Note:** To create and access the Entrust signature, ensure that you have the appropriate Entrust software installed and configured.

## Microsoft CryptoAPI signatures

Microsoft CryptoAPI signatures are a type of digital signature (see “Digital signatures” on page 109). Microsoft CryptoAPI signatures use the Microsoft CryptoAPI signature engine. To use this type of digital signature, it is necessary for the user to obtain a digital certificate.

## Netscape signatures

Netscape signatures are a type of digital signature (see “Digital signatures” on page 109). Netscape signatures use the Netscape encryption engine. To use this type of digital signature, it is necessary for the user to obtain a digital certificate. Netscape signatures use certificates located in the user’s Netscape certificate store.

**Note:** Workplace Forms products do not support the Netscape *browser*, but do support digital certificates provided by the Netscape browser’s NSS certificate store.

## Signature Pad signatures

Signature pad signatures are a blending of electronic signatures and handwritten signatures. You write your signature on a digital pad which captures your handwriting and converts it into an electronic format. This signature is then added to the form, along with a graphic that shows the handwriting. Thus, Signature Pad signatures provide a familiar feel for the signing process.

Most Signature Pad solutions also offer biometric analysis of the signature. This means that the software will examine the signature and compare it to a sample signature that is on file. It will also study specific details of the signature, including how hard the pen is pressed, the angle of the pen against the surface, the writing speed, and so on. If the two signatures match, then the identity of the signer is safely established.

**Note:** Workplace Forms do not support biometric analysis (the comparison of a signature to a stored copy to detect differences between them) of signatures.

These signatures use a Signature Pad that plugs into your computer. The signature pad allows the user to create a handwritten signature that is applied to the form. Before you can use these signatures, you must have the following:

- The Signature Pad extension for the Workplace Forms Viewer
- A signature pad
- Software that enables the signature pad.

The Signature Pad extension is available as a separate install package that adds support for signature pads to the Viewer. Signature pads and their supporting

software are available from a variety of vendors. Workplace Forms Viewer supports the signature pads from Interlink, Topaz, and any WinTab compliant signature pad company.

Signature pad signatures require you to install both hardware and software on the end user's computer. In addition, there is usually a server-side component that stores the sample signatures and compares all new signatures to the samples.

### **When to use Signature Pad signatures**

Signature Pad signatures are more useful than digital signatures; however, they require a certain level of overhead to create and maintain.

In general, large organizations that require legal signatures from their customers, such as banks, are good candidates for Signature Pad signatures.

## **Silanis signatures**

Silanis Technology Inc. provides a signature type that blends a digital signature and a Signature Pad signature. This means that an image of your signature is captured using a Signature Pad, but the actual signature is created using a digital certificate. A digital certificate is a file stored on a disk or other device, such as smart card. You can think of this certificate as a digital pen that you use to create a digital signature. The image of your handwritten signature is then stored as part of the digital signature for later reference. This combines the familiar pen-based signing process with the strength of digital certificates.

### **When to use Silanis signatures**

Silanis signatures are best used in the same circumstances as digital signatures because they are essentially digital signatures with additional features. The decisions between digital signatures and Silanis signatures relies on individual preferences about the usefulness of the handwritten component.

## **Clickwrap signatures**

Clickwrap signatures are electronic signatures that do not require digital certificates. While they still offer a measure of security due to an encryption algorithm, Clickwrap signatures are not security tools. Instead, Clickwrap signatures offer a simple method of obtaining electronic evidence of user acceptance to an electronic agreement. The Clickwrap signing ceremony authenticates users through a series of questions and answers, and records the signer's consent. Clickwrap style agreements are frequently found in licensing agreements and other online transactions. Simply put, Clickwrap signatures gather some information about you and then create a signature that contains that information. For example, a Clickwrap signature might prompt you to specify any or all of the following information:

- Your name
- Personal information, such as your mother's maiden name that can be used to verify your identity
- Repeat a statement to show that you agree with the document you are signing.

After you provide this information, it is then included in the signature itself, and cannot be changed unless the signature is removed completely. Clickwrap signatures provide the same tamper proofing as other signatures, but do not

identify the signer as reliably as other signature types. However, they also do not require any additional infrastructure, such as a PKI system or Signature Pad hardware.

### **When to use Clickwrap signatures**

Clickwrap signatures are useful for the low-value, web-based transactions that they are based on. For example, you might use them to enroll members in a bonus points program.

## **Authenticated Clickwrap signatures**

Authenticated Clickwrap signatures are a blending of Clickwrap and digital signatures. This enables users to securely sign a form without relying on an extended PKI infrastructure. During normal use, the user signs the form by entering an ID and secret, such as a password. When the form is sent to a server, the server retrieves the user's secret from a database and uses that secret to verify the signature. Furthermore, the server can notarize the Authenticated Clickwrap by signing it with a digital certificate, thereby creating a secondary digital signature. This secondary signature shows that the server has confirmed the identity of the signer, and ensures that the original signature can be trusted over time.

Authenticated Clickwrap signatures work like normal Clickwrap signatures, except they also incorporate a shared secret. Typically, this shared secret is a user ID and password. When you sign the form, you provide your shared secret as part of the signature, using the typical Clickwrap question and answer system. When you submit the form, the server then creates a second signature using its copy of the shared secret, and compares it to the signature in the form. If the signatures match, then the server has positively identified you as the signer, and the server then countersigns the form with a digital signature.

This combines the ease-of-use of the Clickwrap signature with the inherent strength of a digital signature and relies on a shared secret infrastructure that likely already exists in the organization.

### **When to use authenticated Clickwrap signatures**

Authenticated Clickwrap signatures are an effective solution for organizations that cannot maintain an extensive PKI infrastructure, but continue to require a high degree of security. Typically, they work best for organizations that currently make use of user IDs and passwords, or some other shared secret.

---

## **Creating signature buttons**

The following procedures outline how to set up each of the signature button types.

**Note:** You cannot place a signature button into a pane or table item's template (that is, the content of the `xforms:group`, `xforms:repeat`, or `xforms:switch`. Although it is possible to do in the Designer, this button will not function correctly in the Viewer. However, a signature button can sign any XFDL, including panes and tables and the data to which their contained controls bind.


## **Creating a Generic RSA signature button**

To create a Generic RSA signature button:

1. Add a button to the form.





For detailed information about creating buttons, see “Creating buttons” on page 68.

2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Generic RSA**.
6. Click **Finish**.
7. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, do the following:
  - a. In the top-right corner of the Properties view, click .
  - b. Click **Show Advanced Properties**.
  - c. In the Properties view, expand **Signature**.
  - d. Click within the **signformat** value field.
  - e. Add the following text to the end of the current **signformat** value:  
`;delete="off"`.

## Creating a Microsoft CryptoAPI signature button

To create a Microsoft CryptoAPI signature button:

1. Add a button to the form. For detailed information about creating buttons, see “Creating buttons” on page 68.
2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Crypto API**.
6. Click **Finish**.
7. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, do the following:
  - a. In the Properties view, click .
  - b. Click **Show Advanced Properties**.
  - c. In the Properties view, expand **Signature**.
  - d. Add the following text to the end of the current **signformat** value:  
`;delete="off"`
8. By default, Microsoft CryptoAPI signature buttons use the Microsoft Base Cryptographic Service Provider. If you want to use a different cryptographic service provider (CSP), do the following:
  - a. Click the Properties view, click .


- b. Click **Show Advanced Properties**.
- c. In the Properties view, expand **Signature**.
- d. Add the following text to the end of the current **signformat** value:
 

```
;csp="yourcsp";csptype="yourcsptype"
```

 where **yourcsp** is the CSP and **yourcsptype** is the type of CSP.

## Creating a Clickwrap signature button

To create a Clickwrap signature button:

1. Add a button to the form. For detailed information about creating buttons, see “Creating buttons” on page 68.
2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Clickwrap**.
6. Click **Next**.
7. Set the **Clickwrap Signature Details**:
  - **Title** — The title of the signing ceremony. This text describes the signing ceremony, the company, or the title of the agreement.
  - **Prompt** — Typically used to explain the signing ceremony to users.
  - **Main Text** — Contains the main text of the agreement. For example, the text of a licensing agreement. You can add as much text as necessary to this parameter. The signing ceremony automatically displays scroll bars if the text is longer than the display field.
  - **Question** — Lets you prompt the user to ask from one to five questions that help establish the identity of the user.
  - **Default Answer** — These are the answers to the questions. To pre-populate the answer fields that are displayed when the user signs the form, enter the answers here. Otherwise, you can leave these fields blank.
8. Click **Finish**.
9. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, do the following:
  - a. Click the Properties view, click .
  - b. Click **Show Advanced Properties**.
  - c. In the Properties view, expand **Signature**.
  - d. Add the following text to the end of the current **signformat** value:
 

```
;delete="off"
```
10. If you want to customize the button, add the following attributes to the **signformat** property using the method described in step 9:
  - **echoPrompt** — Use this to instruct the user to echo the **echoText**. Generally, if you include **echoText**, you might want to include the text: **Please type the following phrase to show that you understand and agree to this contract**.
  - **echoText** — This is the actual text that the user should echo, or re-type. For example, **I understand the terms of this agreement**.



- **buttonPrompt** — This is an instruction line that appears above the **Accept** and **Reject** buttons. The user must click the **Accept** button to sign, so the prompt might read, **Click Accept to sign this document**. The default setting is **Click the Accept button to sign**.
- **acceptText** — Sets the text that the **Accept** button displays. The default text is **Accept**.
- **rejectText** — Sets the text that the **Reject** button displays. The default text is **Not Accept**.

For example, to add an echo prompt, add the following text to the end of the current **signformat** value:

```
;echoPrompt="Please type the following phrase"
```

## Creating an Authenticated Clickwrap signature button

To create an Authenticated Clickwrap signature button:

1. Add a button to the form. For detailed information about creating buttons, see “Creating buttons” on page 68
2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Authenticated Clickwrap**.
6. Click **Next**.
7. Set the **Authenticated Clickwrap Signature Details**:
  - **Title** — The title of the signing ceremony. This text describes the signing ceremony, the company, or the title of the agreement.
  - **Prompt** — Typically used to explain the signing ceremony to users.
  - **Main Text** — Contains the main text of the agreement. For example, the text of a licensing agreement. You can add as much text as necessary to this parameter; the signing ceremony automatically displays scroll bars if the text is longer than the display field.
  - **Question** — Lets you prompt the user to ask from one to five questions that help establish the identity of the user.
  - **Default Answer** — These are the answers to the questions. To pre-populate the answer fields that are displayed when the user signs the form, enter the answers here. Otherwise, you can leave these fields blank.
  - **Signer** — Indicates which answer identifies the signer. **Signer** and **Secret** cannot reference the same answer.
  - **Secret** — Indicates which answer contains the secret. **Signer** and **Secret** cannot reference the same answer.
8. Click **Finish**.
9. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, do the following:
  - a. In the Properties view, expand **Signature**.
  - b. Add the following text to the end of the current **signformat** value:
 

```
;delete="off"
```


10. If you want to customize the button, add any of the following attributes to the **signformat** property using the method described in step 9:
  - **echoPrompt** — Use this to instruct the user to echo the **echoText**. Generally, if you include **echoText**, you might want to include the text: **Please type the following phrase to show that you understand and agree to this contract.**
  - **echoText** — This is the actual text that the user should echo, or re-type. For example, **I understand the terms of this agreement.**
  - **buttonPrompt** — This is an instruction line that appears above the **Accept** and **Reject** buttons. The user must click the **Accept** button to sign, so the prompt might read, **Click Accept to sign this document.** The default setting is **Click the Accept button to sign.**
  - **acceptText** — Sets the text that the **Accept** button displays. The default text is **Accept.**
  - **rejectText** — Sets the text that the **Reject** button displays. The default text is **Not Accept.**
  - **readonly** — Indicates which answers are read-only. This is useful if you pre-populated the answers and want to ensure that certain answers cannot be changed.

For example, to add an echo prompt, add the following text to the end of the current **signformat** value:

```
;echoPrompt="Please type the following phrase"
```

## Creating an Entrust signature button


To create an Entrust signature button:

1. Add a button to the form. For detailed information about creating buttons, see “Creating buttons” on page 68.
2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Generic RSA**.
6. Click **Finish**.
7. Click the Properties view, click .
8. Click **Show Advanced Properties**.
9. In the Properties view, expand **Signature**.
10. In the **signformat** value field, change engine="Clickwrap" to engine="Entrust".
11. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, add the following text to the end of the current **signformat** value: `;delete="off"`

## Creating a Netscape signature button

To create a Netscape signature button:

1. Add a button to the form. For detailed information about creating buttons, see “Creating buttons” on page 68.
2. Right-click the button and click **Wizards** → **Signature Wizard**.


3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Generic RSA**.
6. Click **Finish**.
7. Click the Properties view, click .
8. Click **Show Advanced Properties**.
9. In the Properties view, expand **Signature**.
10. In the **signformat** value field, change engine="Generic RSA" to engine="Netscape".
11. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, add the following text to the end of the current **signformat** value:
 

```
;delete="off"
```

## Creating a Signature Pad signature button

To create a Signature Pad signature button:

1. Add a button to the form. Make sure the button is large enough to display the user’s signature after they sign the form. For detailed information about creating buttons, see “Creating buttons” on page 68.
2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Clickwrap**.
6. Click **Next**.
7. Set the **Clickwrap Signature Details**:
  - **Title** — The title of the signing ceremony. This text describes the signing ceremony, the company, or the title of the agreement.
  - **Prompt** — Typically used to explain the signing ceremony to users.
  - **Main Text** — Contains the main text of the agreement. For example, the text of a licensing agreement. You can add as much text as necessary to this parameter. The signing ceremony automatically displays scroll bars if the text is longer than the display field.
  - **Question** — Lets you prompt the user to ask from one to five questions that help establish the identity of the user.
  - **Default Answer** — These are the answers to the questions. To pre-populate the answer fields that are displayed when the user signs the form, enter the answers here. Otherwise, you can leave these fields blank.
8. Click **Finish**.

9. Click the Properties view, click .
10. Click **Show Advanced Properties**.
11. In the Properties view, expand **Signature**.
12. In the **signformat** value field, change engine="Clickwrap" to engine="SignaturePad".
13. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, add the following text to the end of the current **signformat** value: ;delete="off"
14. If you want to customize the button, add any of the following attributes to the **signformat** property using the method described in step 11:
  - **echoPrompt** — Use this to instruct the user to echo the **echoText**. Generally, if you include **echoText**, you might want to include the text: **Please type the following phrase to show that you understand and agree to this contract**.
  - **echoText** — This is the actual text that the user should echo, or retype. For example, **I understand the terms of this agreement**.
  - **buttonPrompt** — This is an instruction line that appears above the **Accept** and **Reject** buttons. The user must click the **Accept** button to sign, so the prompt might read, **Click Accept to sign this document**. The default setting is **Click the Accept button to sign**.
  - **acceptText** — Sets the text that the **Accept** button displays. The default text is **Accept**.
  - **rejectText** — Sets the text that the **Reject** button displays. The default text is **Not Accept**.
  - **readonly** — Indicates which answers are read-only. This is useful if you pre-populated the answers and want to ensure that certain answers cannot be changed.
  - **tsp** — Defines the preferred signature pad software/hardware to use: **Interlink**, **Topaz**, or **WinTab**.
  - **startText** — Text to display on the button that starts the signature capture.
  - **endText** — Text to display on the button that ends the signature capture.
  - **penColor** — The color to use when drawing the signature on the screen. This is either a color name or a comma separated list of RGB values.
  - **backgroundColor** — The color to use for the background of the signature graphic. This is either a color name or a comma separated list of RGB values.

For example, to add an echo prompt, add the following text to the end of the current **signformat** value: ;echoPrompt="Please type the following phrase"

15. In the Properties view, set **signatureimage** to any value. When the user signs the form, the Viewer will store the image of their signature within the form as an item. The name of the item will be the value of **signatureimage**. You will use the **signatureimage** value in step 17 below.
16. Enclose two images within your form:
  - The unsigned image will be displayed on the signature button when the form is unsigned (that is, before the user signs the form). For example, the image could contain the text, **Click here to sign**.
  - The invalid image will be displayed on the signature button when the signature is invalid. For example, the image could contain the text, **Invalid Signature**.

For detailed information about enclosing images, see “Adding an image file to a form” on page 81.

17. Setup a formula for the button’s **image** property that will determine which image to display on the button:
  - If the **signer** option is empty (meaning the form is unsigned), the button will display the unsigned image.
  - If the **signer** option is invalid (meaning the signature is invalid), the button will display the invalid image.
  - If the **signer** option is anything else (meaning the form is signed), the button will display the **signatureimage**.

For detailed information about formulas, see “Formulas” on page 85.

18. In the Properties view, expand **Appearance**.
19. Set **imagemode** to control how signature images are modified to fit within the button. For detailed information about re-sizing and cropping images on buttons, see “Resizing and cropping images on buttons and labels” on page 82.

## Creating a Silanis signature button

To create a Silanis signature button:

1. Add a button to the form. Make sure the button is large enough to display the user’s signature after they sign the form. For detailed information about creating buttons, see “Creating buttons” on page 68.
2. Right-click the button and click **Wizards** → **Signature Wizard**.
3. Under **What does the signature sign?**, click either of the following:
  - **The complete form** — The signature button will sign the entire form.
  - **Parts of the form** — The signature button will sign part of the form. For detailed information about creating a signature button that signs part of a form, see “Creating a signature button that signs part of a form” on page 122.
4. Click **Next**.
5. Under **What type of signature is it?**, click **Clickwrap**.
6. Click **Next**.
7. Set the **Clickwrap Signature Details**:
  - **Title** — The title of the signing ceremony. This text describes the signing ceremony, the company, or the title of the agreement.
  - **Prompt** — Typically used to explain the signing ceremony to users.
  - **Main Text** — Contains the main text of the agreement. For example, the text of a licensing agreement. You can add as much text as necessary to this parameter; the signing ceremony automatically displays scroll bars if the text is longer than the display field.
  - **Question** — Lets you prompt the user to ask from one to five questions that help establish the identity of the user.
  - **Default Answer** — These are the answers to the questions. To pre-populate the answer fields that are displayed when the user signs the form, enter the answers here. Otherwise, you can leave these fields blank.
8. Click **Finish**.
9. In the Properties View, expand **Signature**.
10. In the **signformat** value field, change engine=“Clickwrap” to engine=“Silanis”.

11. By default, the user can delete a signature after signing a form. If you want to prevent users from deleting the signature, do the following:
  - a. In the Properties view, expand **Signature**.
  - b. Add the following text to the end of the current **signformat** value:  
 ;delete="off"
12. In the Properties view, set **signatureimage** to any value. When the user signs the form, the Viewer will store the image of their signature within the form as an item. The name of the item will be the value of **signatureimage**. You will use the **signatureimage** value in step 14 below.
13. Enclose two images within your form:
  - The unsigned image will be displayed on the signature button when the form is unsigned (that is, before the user signs the form). For example, the image could contain the text, **Click here to sign**. Silanis provides a standard image that you can use.
  - The invalid image will be displayed on the signature button when the signature is invalid. For example, the image could contain the text, **Invalid Signature**.

For detailed information about enclosing images, see “Adding an image file to a form” on page 81.
14. Setup a formula for the button’s **image** property that will determine which image to display on the button:
  - If the **signer** option is empty (meaning the form is unsigned), the button will display the unsigned image.
  - If the **signer** option is invalid (meaning the signature is invalid), the button will display the invalid image.
  - If the **signer** option is anything else (meaning the form is signed), the button will display the **signatureimage**.

For detailed information about formulas, see “Formulas” on page 85.
15. In the Properties view, expand **Appearance**.
16. Set **imagemode** to control how signature images are modified to fit within the button. For detailed information about resizing and cropping images on buttons, see “Resizing and cropping images on buttons and labels” on page 82.
17. If you want to make the signature button a *no-lock signature button* or an *approval signature button*, see “No-lock signatures and approval signatures.”

### **No-lock signatures and approval signatures**

By default, a Silanis signature locks the data that it signs. Thereafter, you cannot make changes to the signed data.

You can also create a *no-lock* Silanis signature button. Clicking this type of signature button signs the form, but does not lock the data that it signs. You can then make changes to the data even though it is signed. Making changes in this way invalidates the signature, but you can then apply a second signature to approve the changes.

For example, your form may include an employee section and a section for the employee’s manager. The employee completes their section and signs the form using a no-lock signature. The manager then reviews the form, makes corrections to the employee section, and then signs the form using an approval signature to approve the changes.



Similarly, your form may include a “public” section and an “office use only” section.

An approval signature must sign all of the data covered by the no-lock signature, including the no-lock signature itself. If not, the no-lock signature will not return to a valid state when the approving signature is applied. The approval signature may also cover additional data.

### Creating no-lock signature buttons

To create a no-lock signature:

1. Select the signature button. For detailed information about creating Silanis signature buttons, see “Creating a Silanis signature button” on page 119.
2. In the Properties view, expand **Signature**.
3. Click within the **signformat** value field.
4. Add to the end of the current value `;lock="off"`.

#### Notes:

- To prevent the no-lock signature from signing its signature image, use the **signitemrefs** filter to omit the entire data item from the signature. If your signature item already contains a **signitemrefs** filter that omits items, add the signature image to the list of items. For detailed information about this filter option, see “Signature filter properties” on page 123.
- To prevent all approving signatures from signing the signature image for the no-lock signature, use the **signitemrefs** filter option. You must add this filter to all signatures that will sign the no-lock signature.

### Creating approval signature buttons

For detailed information about creating an approval signature, see “Creating a Silanis signature button” on page 119.

When you create an approving signature, you must:

- Prevent the no-lock signature from signing its own signature image.
- Prevent all approving signatures from signing the image for the no-lock signature. This ensures that the signature image can change later if the state of the signature changes (for example, if something happens to make the signature invalid, then the image must change to reflect that).

## Signing portions of forms

Forms are frequently signed by more than one person. For example, some forms include a “For Office Use Only” section that requires an additional signature by one of the staff processing the form, in addition to the person submitting the form. In these cases, the office worker must be able to enter more information in the unsigned portion of the form and then add their own signature.

Signature filters specify which parts of the form a particular signature will sign. This means that you can create one signature that signs the entire form, or one that only signs the first portion of the form, and then a second signature that signs the second portion of the form.

**Note:** By default, if you specify a signature button on a form and do not include any signature filters, the entire form is signed.

There are certain form items that you should always exclude from signatures. For example, when you click a **Submit** button, the **triggeritem** option is automatically set, recording the name of the button that triggered the submission. However, if a signature is applied to the **triggeritem** option, the Viewer is unable to update the option correctly.

In general, you should always exclude the following form elements from a signature:

- The **triggeritem** option
- The **coordinates** option
- Any portion of the form that subsequent users will change
- Subsequent signatures and signature buttons
- The signature item that stores the information for the signature you are creating.

## Types of filtering

When creating a signature button that signs part of a form, there are two ways you can control what the signature button will and will not sign:

- You can specify the items that the button *will* sign (leaving the rest of the form unsigned). This is referred to as *keep* filtering.
- You can specify the items that the button *will not* sign (making the rest of the form signed). This is referred to as *omit* filtering.

A form is more secure if you use omit filtering. Doing so will ensure you do not accidentally exclude items that should be signed, and will prevent malicious users from adding to the form's contents without breaking the signature.


If you use keep filtering, the signature button should only be used to sign another signature that uses omit filtering.

## Creating a signature button that signs part of a form

To set up a signature button that signs part of a form:

1. Right-click the button and click **Wizards** → **Signature Wizard**.
2. Under **What does the signature sign?**, click **Parts of the form**.
3. Click **Next**.
4. Under I want to configure the signature by defining, click either of the following:
  - **Items not to sign** — Lets you select the items not to sign
  - **Items to sign** — Lets you select the items to sign
5. Click **Next**.
6. Use the controls to select the *entire pages* that will be **Signed** or **Not Signed** by the signature button.
  - To move all pages from **Signed** to **Not Signed**, click >>>.
  - To move all pages from **Not Signed** to **Signed**, click <<<.
  - To move a page from **Signed** to **Not Signed**, select the page from the list and click >.
  - To move a page from **Not Signed** to **Signed**, select the page from the list and click <.
7. Click **Next**.



8. Use the controls to select the *individual items* that will be **Signed** or **Not Signed** by the signature button.
  - To move all items from **Signed** to **Not Signed**, click >>>.
  - To move all items from **Not Signed** to **Signed**, click <<<.
  - To move an item from **Signed** to **Not Signed**, select the item from the list and click >.
  - To move an item from **Not Signed** to **Signed**, select the item from the list and click <.
  - To select items from the canvas, click , select the items, and then click **Finished** (in the top left corner of your screen).
9. Click **Next**.
10. Select the type of signature:
  - **Generic RSA**
  - **Crypto API**
  - **Clickwrap**
  - **Authenticated Clickwrap**
11. Do either of the following:
  - For **Generic RSA** or **Crypto API** signatures, click **Finish**.
  - For **Clickwrap** or **Authenticated Clickwrap** signatures, click **Next** and set the signature options. (For detailed information about setting Clickwrap and Authenticated Clickwrap signature options, see “Creating a Clickwrap signature button” on page 114 and “Creating an Authenticated Clickwrap signature button” on page 115.) Then click **Finish**.

You can further customize the filtering of signature buttons by modifying the signature properties. For detailed information about signature filtering properties, see “Signature filter properties.”

### Signature filter properties

Complex forms frequently require sophisticated filtering that you can setup by editing the following signature properties directly. Signatures can include any number filters.

**Note:** When using signatures, regardless of the signature filters in use, the following rules apply:

- The `mimedata` option in a signature item is always omitted from the signature that item represents.
- The `mimedata` option in a data item that stores a signature image (see the `signatureimage` option) is always omitted from the signature that image represents.

Signature filters are applied with an order of precedence so that filter properties are always processed in a consistent manner. The following table lists the behavior of the filter properties and the order in which the Viewer applies them. For detailed descriptions of properties, see “Appendix B: Options” on page 187.

Order	Property	Behavior if using the <b>Omit</b> Filter	Behavior if using the <b>Keep</b> Filter	Notes

1.	signinstance	Omits only data in the indicated instance; throws them out.	Keeps only data in the indicated instance; throws others out.	
2.	signnamespaces	Omits only elements and attributes in the namespaces indicated; throws them out.	Keeps only elements and attributes in the namespaces indicated; throws others out.	An element is kept if any of its children are kept, even if it is in the wrong namespace.
3.	signitems	Omits only the types listed; omitted items are not signed.	Keeps only the types listed; all other types are not signed.	
4.	signoptions	In the items that remain (not omitted by signitems) omits all listed options. Omitted options are not signed.	In the items that remain, keeps all indicated options. All other options remain unsigned.	
5.	signpagerefs	Omits the specified pages. Overrides settings in signitems and signoptions.	Keeps the specified pages. Respects settings in signitems and signoptions.	Omitted pages are not completely deleted; the page sid is preserved.
6.	signdatagroups and signgroups	Omits the items in that group, even if they are of a type that should be kept according to a signitems setting.	Keeps the items in that group, even if they are of a type that should not be kept according to a signitems setting.	These settings override those in signpagerefs.
7.	signitemrefs	Omits the specified items and overrides previous settings.	Keeps the specified items and overrides previous settings. Respects settings in signoptions.	These settings override signitems, signgroups, signpagerefs, and signdatagroups.
8.	signoptionrefs	Omits the specified options, overriding any previous settings.	Keeps the specified options, overriding any previous settings. If the item containing the option has been omitted, that item's sid and the specified option are preserved.	This option's setting overrides all other filter options.

---

## Specifying the display for a signature button

Signatures buttons can change between two states: unsigned and signed. When a signature button is unsigned, it is generally good practice to have the button display instructions, such as **Click here to sign**. When a signature button is signed, it is generally good practice to show the identity of the person who signed the form.

Since buttons display the text in their **value** option, you must set the value option, depending on whether the button is unsigned or signed. To do this, you must use a formula. The Compute Wizard relies on the button's **signer** option. The **signer** option is created by the Viewer when the button is signed. This means that there is no **signer** option when the button is unsigned. If the **signer** option has no value (it does not exist), then there is no signature and the button should read **Click here to sign**. If the signer option has a value, then there is a signature and the button should show the identity of the signer. Since the **signer** option stores the signer's identity, you can set the button's value to equal that of the **signer** option.

To set the button's text (value):

1. In the Properties view, expand **General** and **value**.
2. Click within the **compute** value field to highlight the contents. By default, the Compute value displays the following initial code for the selected signature button:

```
signer== ''?' : signer
```

3. Change the code to the following:

```
signer==''? 'Click to sign' :signer''
```

---

## Making a signature button mandatory

Most signatures are required on documents; they often include a place to sign and are not accepted or processed without a signature. Similarly, you may want to make your signature buttons mandatory. When you make the signature button mandatory, the Viewer prevents users from submitting the form until they have signed it to ensure that you do not receive unsigned forms. To ensure that a user signs a form before submitting it, you can make a signature button mandatory.

To make a single signature button mandatory:

1. Select the signature button.
2. In the Properties view, expand **Format**, **format**, and **constraints**.
3. Set **mandatory** to **on**.

---

## Signature properties

Signature properties are grouped under **Signature** in the Properties view. You must select **Show Advanced Properties** to display signature properties. For detailed information about displaying Advanced Properties, see "Showing advanced properties" on page 53.

For detailed information on specific properties, see "Appendix B: Options" on page 187.



---

## Web services

Web services let one application to talk to another server using the Internet. For example, using Web services you can create functions that let the Viewer communicate with server-side applications and update information in a form without the user having to submit the form.

The Designer supports Web services by letting you embed any number of Web Services Definition Language (WSDL) documents in your form. Once embedded, you can access the document's functions as though they were XFDL functions. For detailed information about WSDL, see the *Workplace Forms XFDL Specification* document.

**Note:** You must have a WSDL document ready before attempting to add Web services to a form. For detailed information about WSDL, see <http://www.w3.org/TR/wsdl>.

---

### Adding Web services to a form

Before you can add Web services to your form, you must have a complete WSDL document that defines the Web services you want to provide. You can get more information about WSDL documents at the W3C web site.

**Note:** Web services must not include the underscore character ( \_ ) in either service or port names, but can include underscores in operation names.

To add Web services to your form:

1. In the Enclosures view, expand **WSDL**.
2. Right-click **WebServices** and choose **Enclose WSDL File**.
3. In the Choose File window, browse to the WSDL file you want to enclose and then click **Open**.

Now, you can generate instances from the WSDL file. For detailed information about generating instances from a WSDL file, see "Creating instances from WSDL messages" on page 134.

---

### Deleting an enclosed WSDL file from a form

Once you have finished creating data instances from a WSDL file, you should delete the enclosed WSDL file to decrease the size of your form, thereby improving performance.

To delete a WSDL file from the form:

1. In the Enclosures view, select the WSDL file you want to delete.
2. Right-click and choose **Delete**.
3. Repeat steps 1 through 2 to delete additional WSDL files.



---

## XForms

XForms creates a separate data layer inside the form, letting you collect data from the form and copy it into a separate block of XML that you can format as you like.

Separating the data layer from the presentation layer makes XForms device independent. The data model can be used for all devices. The presentation can be customized for different user interfaces, like mobile phones, handheld devices, and Braille readers for the blind.

Since XForms is device independent and based on XML, it is also possible to add XForms elements directly into other XML applications like VoiceXML (speaking web data), WML (Wireless Markup Language), and SVG (Scalable Vector Graphics).

**Note:** To effectively use XForms you must already be familiar with XForms and have a thorough understanding of XML, data models and XPath. As well, this document does not explain everything about XForms. For detailed information about XForms, see <http://www.w3.org/MarkUp/Forms/>. For detailed information about how XFDL is used in combination with XForms, see the *Workplace Forms XFDL Specification* document.

### Using XForms in the Designer

To use XForms, you use the Designer to define the following:

- **XForms model** — The *data layer* that describes the form's data and logic. For detailed information about XForms model, see "The XForms model" on page 130.
- **XForms user interface (UI) items** — The *presentation layer* that lets the user input data that will be stored in the data layer. You can also use XForms UI items to display data from the XForms model. For detailed information about XForms UI items, see "XForms user interface" on page 138.

You bind the XForms model to the XForms UI items. For detailed information about XForms binding, see "XForms binding" on page 161.

Finally, you must define the data that will be submitted. For detailed information about XForms submissions, see "XForms submissions" on page 165.

### When to use XForms

You can use XForms in any of the following scenarios.

#### XML Applications

XForms is most useful when integrating Workplace Forms with applications that already use XML, especially if those applications already offer XML interfaces. In these cases, you can design forms that will submit the XML data directly to the application, without needing to program a custom module that extracts the data from the form. Furthermore, you can format the data to match any schema, and validate the data against the schema before submission.

#### Non-XML Applications

Even if an application does not use XML, you can still benefit from using

XForms. The XForms model simplifies copying information from one page to another, making wizard-style forms easier to create and manage. Furthermore, although custom programming is still required for back-end processing, the data model makes it far easier to extract data from a form.

---

## Adding XForms support

Adding XForms support creates a default XForms model to which you can add XForms instances, binds, and submissions.

When you create a new form, or open a non-XForms form, by default you do not have access to any of the XForms elements. To work with XForms elements, you must first enable XForms support.

You can add XForms support to a new or existing form.

### Adding XForms support to a new form

To add XForms support to a new form:

1. Click **File** → **New** → **New Workplace Form** to open the New Workplace Form window.
2. Select the project folder.
3. Type a **File name**.
4. Click **Next**.
5. Within the **Choose Template** field, click **Default Empty Form - XForms**.
6. Click **Finish**.

XForms elements are now available for you to use.

### Adding XForms support to an existing form

When you add XForms support to an existing form that already contains XFDL items, it is important to note that the XFDL items are not automatically converted to XForms items.

To add XForms support to an existing form:

In the XForms view, right-click **No Model Exists** and click **Add XForms Support**.

An XForms model is added to the form and XForms elements are now available for you to use.

Tip: If the XForms view is not open, click **Windows** → **Show View** → **XForms**.

---

## The XForms model

The XForms model is a block of XML data that contains three core parts that work together to create a complete model:

- **Data Instances** — Data instances are arbitrary blocks of XML. A data model may contain any number of data instances, and each instance is normally created to serve a particular purpose. For detailed information about XForms data instances, see “XForms data instances” on page 131.
- **Binds** — The data layer and the presentation layer are connected by binds. For detailed information about XForms binding, see “XForms binding” on page 161.



- **Submissions** — Each data instance may have an associated set of submission rules. These rules control how a data instance is transmitted when it is submitted for processing. This is an optional feature, and is only necessary when you want to submit the data instance by itself, without the rest of the form. There are many cases in which you may want to submit the entire form, and then retrieve the data instance from the form during processing. This is particularly true when you are using signatures on your forms. For detailed information about signatures, see “Signatures” on page 107. For detailed information about XForms submissions, see “XForms submissions” on page 165.

**Note:** It is recommended that a form contain only one XForms model, but multiple models are allowed (though they have no ability to interact).

## Naming an XForms model

The form’s first XForms model is the default model. A single model does not require a name. However, if you are adding multiple XForms models, each must have a unique name.

To name an XForms model:

1. In the XForms view, expand **XForms**.
2. Click the model you want to name.
3. In the Properties view, expand **Identification**.
4. Click within the **id** value field.
5. Type the model name and press Enter to set the model name.

---

## XForms data instances

An XForms data instance defines the XML template for the data that will be collected from the form. A data instance can be used to store input values, pre-populate fields with data, or dynamically generate list selections.





An XForms model can have more than one data instance. For example, one data instance can contain user information for a submission while another data instance can contain user preference data. Additionally, you can link each data instance to a button on the form that will trigger the submission of that instance.

It is a good idea to create each data instance, along with its associated binds and submission rules following these steps:

1. **Create a data instance** — The first stage is to create a data instance. In this stage, you model your data instance by adding elements, attributes and text values to the data instance.
2. **Bind the data instance to the form** — The second stage is to bind the data instance to the form. This maps individual data elements to one or more user interface items, so that they share data. For detailed information on binding, see “XForms binding” on page 161.
3. **Set the submission rules** — Finally, you must define the submission rules for the instance if you intend to submit the data separately. These rules determine what data is selected and sets other submission-related properties. For detailed information on submissions, see “XForms submissions” on page 165.

## Creating an XForms data instance

The Instance view has four button commands that you can use to create an instance:

-  **Create new data Instance from the current document** — Use this method if you have already created the visual, or presentation, layer of the form. For detailed information on creating an instance from the current document, see “Creating a data instance from the current document.”
-  **Create a new empty Instance** — Use this method if you want to begin designing a form with the data layer. For detailed information on creating an empty instance, see “Creating an empty data instance” on page 133.
-  **Create a new Instance from a schema** — Use this method if you want to base the form on a schema. This method is also known as schema-based form design. For detailed information on schema-based form design, see “Schema-based form design” on page 133.
-  **Create a new Instance from a WSDL Message** — Use this method if you want to base the instance on an enclosed WSDL message. For detailed information on creating an instance from a WSDL message, see “Creating instances from WSDL messages” on page 134.


You can use any of these methods to create an instance. Ultimately, the method you choose will depend upon your form design approach and what resources you have.

### Creating a data instance from the current document

When you create a data instance from the current document, a data instance is generated based on the XFDL and XForms user interface items displayed in the form. The resulting data instance mirrors these user interface items. This is a quick way of creating a basic data instance after you have completed the user interface layer of the form. You can then use the auto-generated instance as is or modify it as you like.

**Note:** If your form already has a data instance, you will not be able to create a data instance from the current document.

To create an instance from the current document:

In the Instance view, click  **Create a new data Instance from the current document.**

A data instance mirroring the structure of the current form is generated. You can view the instance in the Instance view.

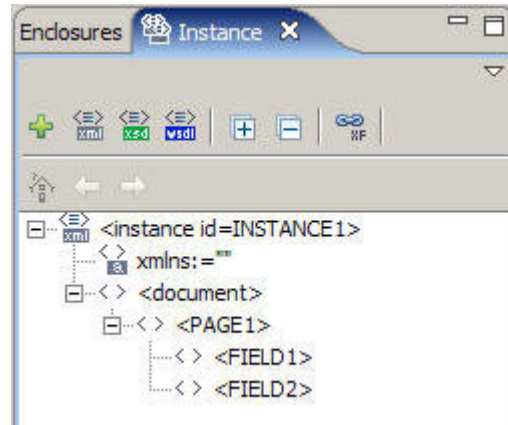
**Note:** The data instance nodes are automatically bound to the generated user interface elements. You cannot create a data instance from the document when there is an existing data instance. Doing so would break all existing binds. If you want to create a new data instance based on your current document, you must first delete existing data instances.

For example, a simple form has one page and two fields. The root level of the form is represented by the document node that contains both a global node and a node

for each page. The page nodes contain the user interface item nodes. For detailed information about nodes, see “Building XForms data instances” on page 134.

Each user interface item’s scope identifier (**sid**) defines the name of the node in the instance.

The generated instance will be the following:

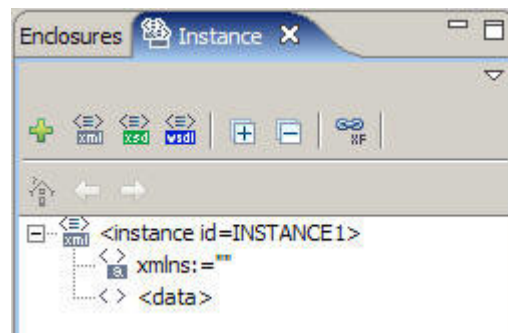


## Creating an empty data instance

To create an empty data instance:

In the Instance view, click **Create a new empty data Instance**.

In the Instance view, the following default data instance is created:



An empty data instance is created with an undefined namespace attribute and a <data> element. The <data> element is the root node of the data instance.

Next, you must build the data instance manually. For detailed information on building a data instance, see “Building XForms data instances” on page 134.

## Schema-based form design

If a schema is enclosed in a form, you can create a prototypical instance that adheres to its rules.

To create a data instance from an enclosed schema:

In the Instance view, click **Create a new Instance from a schema**.

The instance is displayed in the Instance view.

If the form already has XForms items, you can bind them to nodes in the data instance. For detailed information about binding, see “XForms binding” on page 161.

## Creating instances from WSDL messages

Once you have added a WSDL file to your form, you can use its messages to create instances.

To create an instance from a WSDL message:

1. In the Instance view, click  **Creates a new Instance from a WSDL Message.**

The WSDL Message window is displayed, listing the available WSDL messages from which you can create an instance.

**Note:** If your form has more than one XForms model, you must select the model that you want to add the new WSDL-based instance to.

2. Click the message or messages you want to use to create an instance and then click **OK**.

A separate instance is generated for each message you chose.

## Naming a data instance

If you have more than one data instance in your form, you must give a unique name to each instance. In a large form with multiple models and data instances, giving meaningful names to the data instances reduces confusion.

**Note:** Instance names cannot contain special characters (such as spaces, <, >, &, and so on.)

To name a data instance:

1. In the Instance view, click the data instance you want to name.
2. In the Properties view, expand **Identification**.
3. Click within the **id** value field.
4. Type the data instance id and press Enter.

## Building XForms data instances

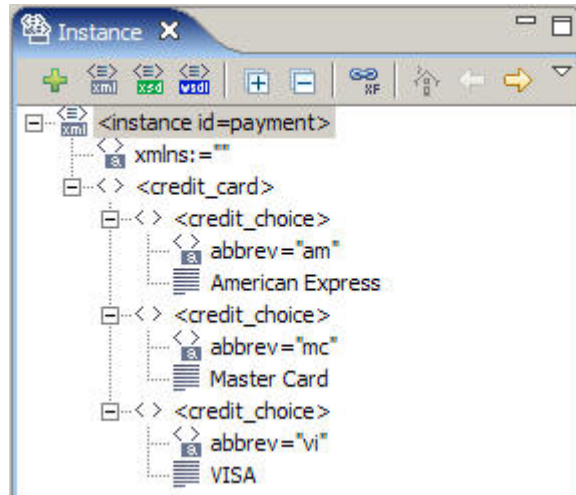
You build a data instance by adding elements to it. You can further define your data instance by optionally adding attributes and text values to elements.

### Nodes

The data instance’s elements and attributes are collectively referred to as *nodes*. Since you will have to bind the form’s user interface items to the various nodes in a data instance, it is important to understand node terminology.

### Node terminology

The following data instance will be used as an example to describe the various node types.



- **root node** — The *root node* is the single node that contains all other nodes. You build your data instance by adding elements and attributes to the root node. A data instance can only have one root node. In the example, the root node is `<credit_card>`.
- **parent node** — The *parent node* is an element that has other elements added to it. When a parent node contains more than one child element, it is also referred to as a *node set*. In the example, a parent node is `<credit_card>`.
- **child node** — A *child node* is an element that is added to a parent element. In the example, `<credit_choice>` is a child node of `<credit_card>`.
- **sibling node** — When you add more than one child to a parent, each child node becomes a *sibling node*. You can add elements to a parent element as children or as siblings. In the example, a sibling node is `<credit_choice>`.

**Note:** Typically, you would only add a sibling element to the data instance if you forgot to add an element or the model has changed and needs to be updated.

- **attribute node** — You can add attributes to an element. An *attribute* extends the functionality of an element and is typically used in the following circumstances:
  - You want to limit the size of data you will be storing.
  - You have a group of items that you want to present to the user as a list of items.
  - You want to submit additional data related to the data entered by the user. For example, you could add an attribute to hold a product id that the user never sees but is submitted when the user picks the product.

In the example, the attribute node is `abbrev`.

**Note:** Node names cannot contain special characters (such as spaces, `<`, `>`, `&`, and so on.)

## Node text values

A node can also have a text value.

- **element text value** — You can add a text value to a child element. Once you bind an element that has a text value to a user interface item, the text value is displayed to the user in the bound user interface item. In the example, the element text values are: American Express, Master Card, VISA.

**Note:** Once you add a text value to an element, it can no longer be a parent node. In other words, you cannot add an element to an element that has a text value.

- **attribute text value** — You can add a text value to an attribute. Once you bind an attribute that has a text value to a user interface item property, the text value is what will be submitted as the item's value property. In the example, the attribute text values are: "am", "mc", "vi". If 'Master Card' is chosen by a user from the credit card list, 'mc' is submitted as the node value instead of 'Master Card'.

## Adding child elements

To add a child element to the parent element:

1. In the Instance view, click the parent element.
2. Right-click and click **Add Element**.  
A new element is created beneath the parent element. If the parent already has children, then the new child is added after the existing children.
3. Double-click the new element, name it and then press Enter.

## Adding sibling elements

Adding a sibling element inserts an element before an existing element. Typically, you would only add a sibling element to the data instance if you forgot to add an element or the model has changed and needs to be updated.

**Note:** Siblings can only be added before an existing child element.

To add a sibling element:

1. In the Instance view, click a child element.
2. Right-click and click **Insert Element Before**.  
A new sibling element is created above the selected child element.
3. Double-click the new element, name it and then press Enter.

## Renaming elements

To rename an element:

1. In the Instance view, double-click the element you want to rename.
2. Type the new element name and then press Enter.

**Important:** If you have already bound an element you want to rename, you will have to update the bind because the XPath is not automatically regenerated. Otherwise, there will be an error.

## Removing elements

To remove an element:

1. In the Instance view, click the data element you want to delete.
2. Do either of the following:
  - Press **Delete**.
  - Right-click the element and click **Delete**.

## Adding a text value to an element

You add a text value to an element when you want to display a value in a user interface item that is bound to the element. For example, if you want to inform a user that an input field is to be used for typing their last name, you could add a

text value of "Type your last name here." to the <last\_name> element. Once the <last\_name> element is bound to the input field, the text value appears in the field.

To add a text value to an element:

1. In the Instance view, right-click the element that you want to add a text value to.
2. Click **Add Text**.
3. Double-click **text**, type the text value and then press Enter.

After you bind the element to an XForms user interface item, the text value appears in the bound user interface item. For detailed information about binding, see "XForms binding" on page 161.

## Adding attributes to an element

An attribute extends the functionality of an element. Essentially, an attribute is another tool you can use to model your instance. For example, you could add three child elements to a <name> parent element: <first>, <middle> and <last>. Alternatively, you could add three attributes to a <name> element: first, middle and last.

You can add one or more attributes to an element. If an element has more than one attribute, each attribute must have a unique name.

To add an attribute to an element:

1. In the Instance view, right-click the element that you want to add an attribute to.
2. Click **Add Attribute**.  
An attribute is added immediately beneath the element.
3. Double-click the newly-added attribute, type a descriptive name for the attribute and then press Enter.
4. To add additional attributes, repeat steps 1 through 2.

## Adding a value to an attribute

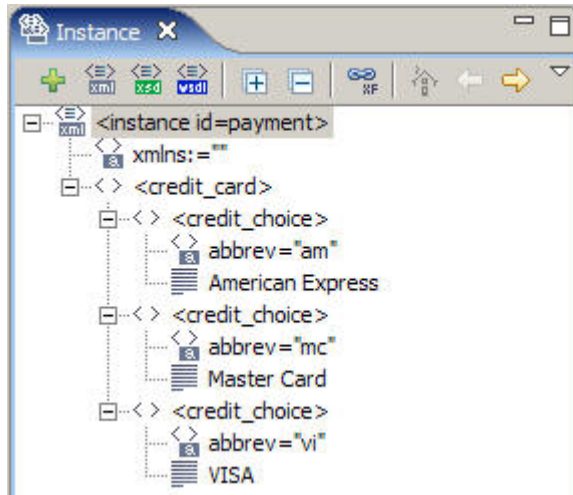
When you add a value to an attribute it creates an optional piece of data that is associated with the node. This value can then be submitted. This is helpful when you want to store a truncated value coming from the user interface or to submit information that is associated with the user-selected data such as an internal product code.

To add a text value to an attribute:

1. In the Instance view, right-click the attribute that you want to add a text value to and click **Edit**.
2. Type the text value and then press Enter.

In the following scenario, each <credit\_choice> element that can be bound to a user interface item has a two-letter **abbrev** attribute. When the user selects a credit card type on the form, the value that is submitted will be two letters long.





When you bind the element to an XForms user interface control, the attribute's value is what will be stored as the element value when a user chooses a value.

### XPath and nodes

XPath referencing lets you bind the node or node set with XForms items in the form, or with model item properties that calculate values or place limitations on the nodes. For detailed information about binding, see “XForms binding” on page 161.

---

## XForms user interface

On its own, the XForms model has no graphic components (user interface) or any way to interact with users. This means that the XForms model must always be contained within a host language that provides a user interface. The Designer supports building XFDL documents that wrap XForms.

**Note:** The topics that comprise this section assume that your form is XForms enabled and that you have already created and named your data instance.

## XForms items

An *XForms item* is an XFDL-wrapped user interface (UI) item that you can bind to a node in the data model or trigger an action based on how the user interacts with the form.

The method you use to add an XForms item to a page is identical to adding an XFDL item: on the Palette you click an XForms item and then click the canvas where you want to place the item.

Since the XForms item is wrapped in XFDL, you can use all the XFDL properties — such as *itemlocation*, and *appearance* — to alter and customize the XForms item as desired.

### XForms item properties

When you examine the XForms item's properties in the Properties view, you can see what differentiates an XForms item from an XFDL item. Aside from having XFDL item properties such as a **sid**, **itemlocation**, and **appearance** properties, the XForms item also has a set of **XForms** properties. It is the **XForms** properties that



define how the XForms item interacts with the form's data layer.

## Adding and binding XForms items - Quick reference

**Note:** This section assumes that you have already modelled the data layer of the form.

XForms can be very complex. To reduce the confusion of adding XForms items to your form and then binding them to the form's data layer, you should follow these steps:

1. In the **Palette**, expand the **Standard Library**.
2. Click the XForms item and then add it to the canvas.
3. Position the item on the form.
4. In the Properties view, expand **General**.
5. If your form is complex, you should name the item by giving the XForms item a descriptive **sid** value. Alternatively, you can accept the auto-generated value.
6. If you want to add a label to the item, type a **label** value.
7. In the Instance view, drag the data node you want to bind to the XForms item.
  - This binds the item to the data layer using the item's **ref** property. To see this bind, in the Properties view, expand **XForms (option)**. For detailed information on XForms binding, see "XForms binding" on page 161.

## Creating XForms labels

Labels are normally used as titles or to identify or describe another XForms item that requires user input, such as fields or check group items.

Since a label requires no user input, most of the time you can use an XFDL **Label** item because it offers more flexibility for appearance and positioning. However, unlike XFDL labels, an XForms label can get its content from a data node.

There are two kinds of XForms labels:

- A standalone text or image label.
- An XForms control label that describes a node in the data instance.

Standalone text or image labels are created using the **Label (Output)** XForms item, while the XForms item labels are created by binding the XForms item's **label** properties to a node in the data instance.

## Setting the XForms item's label property

The **label** property is a mandatory property for most XForms items. It is intended to provide text for the XForms item's label. However, if you prefer to use an XFDL **Label** item to label the XForms item, the content of the XForms item's **label** property may be left blank. If the XFDL label property for the item has a value, it overrides the XForms label property.

To set the XForms item's label property:

1. Select the XForms item that you want to add the label property to.
2. In the Properties view, expand **General**.
3. Click within the **label** value field.
4. Type the label name for the item and press Enter.

## Creating standalone XForms labels

To create an XForms standalone label:

1. In the Palette, expand **Standard Library**.
2. Click **Label (Output)** and then add this item to the form.
3. In the Properties view, expand **XForms (Output)**.
4. If you want to modify the label's appearance and position, use the Properties view to set the appearance and position properties. This could include the label's size, color, font, and positioning.
5. In the Instance view, drag the data node you want to bind to the label.  
The label is bound to the data node using the item's **ref** property.

## Binding an XForms item label to a data node

Every XForms item has a set of label properties that lets you bind the XForms item's label to a node in the data instance. When the form is viewed in the Workplace Forms Viewer, the content of the node is displayed as the XForms item's label.

**Note:** When you bind a label to a node, it overrides the item's default **label** property (found in the **General** properties).

To bind an XForms item label to a data element:

In the Instance view, click the data node you want to bind to the item's label and drag it to the label item.

The label is bound to the data node using the item's **ref** property.

## XForms fields

You use fields to collect information from the user, such as names, dates, dollar amounts, and so on. You can set up fields to check and restrict users entries, to flag errors and omissions and provide help on how to correct them, to format user input in a standard style, and to perform calculations and make logical decisions.

### Creating single line fields

You can use the **Field (Input)** to add a single line field to your form.

**Note:** While this procedure uses **Field (Input)**, you can use the following steps to configure any **input** option type control.

To create a single line field:

1. In the Palette, expand **Standard Library**.
2. Click **Field (Input)** and then add this item to the canvas.
3. If you want to modify the field's appearance and position, use the Properties view to set the appearance and position properties. This could include the field's size, color, font, and positioning.
4. In the Instance view, drag the data node you want to bind to the input field on the canvas.

The input field is bound to the data node using the item's **ref** property.

## Creating multi-line fields

A multi-line field is not limited in the number of lines it can collect and display. You use the XForms **Field (TextArea)** item to create a multi-line field.

To create a multi-line field:

1. In the Palette, expand **Standard Library**.
2. Click **Field (TextArea)** and then add this item to the canvas.
3. If you want to modify the field's appearance and position, use the Properties view to set the appearance and position properties. This could include the field's size, color, font, and positioning.
4. In the Instance view, drag the data node you want to bind to the multi-line field on the canvas.

The multi-line field is bound to the data node using the item's **ref** property.

## Creating password fields

A password field lets a user enter a single line of write-only data. Any text typed in a password field is displayed as a line of asterisks, preventing others from seeing the user's password.

To create password field, you must use the XForms **Field (Secret)** item.

**Note:** Information typed into an XForms **Field (Secret)** item is stored as plain text so it does not provide security.

To create a password field:

1. In the Palette, expand **Standard Library**.
2. Click **Field (Secret)** and then add this item to the canvas.
3. If you want to modify the field's appearance and position, use the Properties view to set the appearance and position properties. This could include the field's size, color, font, and positioning.
4. In the Instance view, drag the data node you want to bind to the password field on the canvas.

The password field is bound to the data node using the item's **ref** property.

## Creating fields from instances

To create a field with a label from an instance:

In the Instance view, drag the instance element on the canvas. The label text is the name of the element.

## XForms lists

A list displays a group of related list items as list choices and lets a user choose either one or more list choices.

### Lists and form design considerations

The type of list you add to your form will depend on how many list items you want the user to choose (**select** or **select1**) and what type of user interface items

your users are used to working with, as well as any space limitations on your page. For example, a popup list takes up less space than does a group of check boxes or radio buttons.

The following are some general suggestions for list design:

- **Check boxes** — Use a **CheckGroup (Select)** to display all list choices to a user and let the user select one or more check boxes.
- **Radio buttons** — Use **RadioGroup (Select1)** if you:
  - Want the user to select only one choice.
  - Want to show all of the possible list choices at once.
  - Want the user to have to click just once to select the list choice.
  - Have a limited number of list choices.
- **Lists** — Use **List (Select1)** lists if you:
  - Want to limit the user to select only one list choice.
  - Want to save space on your form.
  - Have your form dynamically generate list choices from the data model and insert them into the form.

Additionally, you can choose from three kinds of lists:

- **Popup List** — A popup list appears as a single field on your form. It allows the user to select a single choice.
- **Combo box list** — A combo box list appears as a single field on your form. It allows the user to type in a choice or choose one from a popup list.
- **Box list** — A box list can be any size and displays list items in a scrolling list.

The type of list you use will depend on the user data you want to collect from the list and form design considerations.

## Creating XForms lists

You can use the Designer to create XForms lists using one of the following methods:

- **Displaying the list items from a data node** — The list items are dynamically generated from a data instance node set. There is an associated data node for each choice in the list. For detailed information about displaying list items from a data node, see “Displaying list items from a data node set.”
- **Defining list items in the user interface** — In this type of list, you create individual list items in the select or select1 list. This type of list is associated to a single data node that stores the user’s selection. For detailed information about defining list items in the user interface, see “Defining list items in the user interface” on page 144.

## Displaying list items from a data node set

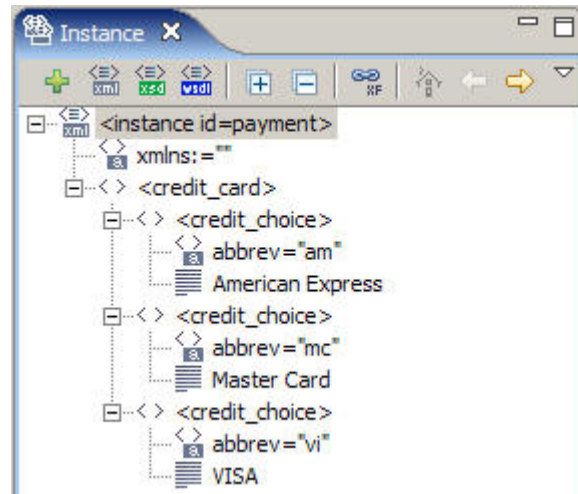
You can display list choices from a node set in the data instance. When you use this method to create a list, the child nodes are dynamically displayed as list items. The advantage to using this method to display list choices is that if you need to update the list, you only have to update the data instance.

While this procedure uses an XForms **List (Select1)** item as an example, you can also apply this procedure to the following XForms items:

- **List (Select)**
- **Combobox (Select1)**

- **Popup (Select1)**

To display list choices from a node set, you must have a node set (parent node) with child nodes that represent each list choice to be displayed in the list. The following sample data instance will be used.



To add a list to the form:

1. Add a **List (Select1)** to the canvas.
2. In the Properties view, click within the **label** value field.
3. Type the label for the list and press Enter.

Next, you must set the **List (Select1)** item's **nodeset** property which binds the list item to the data element node set.

#### **Binding an XForms list to a node set:**

To bind the list to a node set:

1. In the Instance view, right-click the node that contains the nodes you want to display as list choices and click **Copy Reference**. From example data instance, you would select `<credit_choice>`.
2. On the canvas, click the XForms **List (Select1)** item.
3. In the Properties view, expand **XForms (select1)**, and **itemset**.
4. Click within the **nodeset** value field.
5. Right-click and click **Paste**.

Next, you must set the **itemset**, **label**, **ref** property. This property binds the data nodes to the list, displaying them as list items.

#### **Displaying list items in an XForms list:**

To display the list items in the list:

1. On the canvas, click the XForms **List (Select1)** item.
2. In the Properties view, expand **XForms (select1)**, **itemset**, and **label**.
3. Click within the **ref** value field.
4. Type `.` (period)

Defining the **ref** value as `.` (period) displays all the text values of the data nodes defined in the **nodeset** property.

**Note:** If the data node uses an attribute to define its value, the XPath reference would be: *@attribute*

You must now define the list's **value** property. The **value** property is the user's selection from the list.

#### **Setting the value selected from an XForms list:**

To set the value selected from the list:

1. On the canvas, click the XForms **List (Select1)** item.
2. In the Properties view, expand **XForms (select1)**, **itemset**, and **value**.
3. Click within the **ref** value field.
4. Type the path of the node that will store the chosen list item value.  
Almost always, the **ref** value will be . (period). In this example, you would set **ref** to **@abbrev** to store the credit card's abbreviation instead of the full credit card name.

Next, you must define where the value will be stored in the data layer.

#### **Storing the XForms list choice in the data layer:**

To store the XForms list choice in the data layer:

1. On the canvas, click the XForms **List (Select1)** item.
2. In the Properties view, expand **XForms (select1)**.
3. Click within the **ref** value field.
4. Type the XPath of the data element that will store the user's list item choice.

#### **Defining list items in the user interface**

You can define and display list choices in the XForms. Defining your list choices in the **Choice (Item)** lets you minimize the number of nodes in your data instance. Instead of creating data nodes, you add an item to the list for every list choice you want to offer. The XForms **Choice (Item)** is then associated with a single data node that records the user's selection.

You can define list choices in the user interface using XForms Choice (Item).

For detailed information about adding items to a check group or radio group, "Creating a list using XForms CheckGroup."

For detailed information about adding items to a list, "Creating a list using List (Select1)" on page 146.

#### **Creating a list using XForms CheckGroup**

You use the XForms Select or Select1 CheckGroup item to present list items to the user as check boxes.

While this procedure uses **CheckGroup (Select1)** as an example, you can also apply this procedure to the following XForms items:

- **RadioGroup (Select1)**

To create a check group list and bind it to the data node that will store the user's choice:

1. Add a **CheckGroup (Select1)** to the canvas.

2. In the Instance view, click the data node that will store the user's choice from the list and drag it to the **CheckGroup (Select1)** .  
The **CheckGroup (Select1)** is bound to the data node.  
Since you are not linking the individual list choices to data nodes, you must now turn off the **itemset** property.
3. Click the **itemset** property value button to turn it off.

Next, you must add one or more check group items to the check group. Each check group item represents a list choice.

#### **Adding check group items to an XForms CheckGroup list:**

You add **Choice (Item)** to a **CheckGroup** to present the user with check box list items.

While this procedure uses **Choice (Item)**, you can also use this procedure to:

- add **Choice (Item)** to a **RadioGroup**

To add check group items to a **CheckGroup ( Select1)**:

1. Add a **Choice (Item)** to the **CheckGroup (Select1)**.

**Note:** If you created a **RadioGroup (Select1)** instead of a **CheckGroup (Select1)**, you add one or more **Choice (Item)**.

2. In the Properties view, expand **XForms (item)**, and **label**.
3. Click within the **Text** value field.
4. Type the label text for the check box and press Enter.
5. If you want to set the item's value property, see "Setting a list item's value property."
6. Repeat steps 1 through 4 to add and label the remaining check boxes to the check group.

#### **Setting a list item's value property:**

To set a list item's value property, you must first add an XForms list to the canvas and then add items to the list.

You can optionally set a list item's **value** property. This property lets you define the value that is passed to the form's data layer. Typically, you would set this property if the item label is long and you want to save storage space. For example: an item with a label of 'American Express' could have a value of 'AMEX'.

**Note:** You can apply this procedure to all XForms list items.

To set an item's **value** property:

1. On the canvas, click the XForms list.
2. In the Properties view, expand **XForms (select) or (select1)**, and **Item**.  
The list's items are displayed.
3. Expand the **item** property of the item whose **value** property you want to set.
4. Click within the **Text** value field.
5. Type the text and press Enter to set the **value**.

## Creating a list using List (Select1)

You can create a **select** or **select1** list and then add and define the list items to the list.

While this procedure uses **List (Select1)** as an example, you can also apply this procedure to the following XForms items:

- **List (Select)**
- **Popup (Select1 )**
- **ComboBox (Select1)**

To create a **List (Select1)**:

1. Add a **List (Select1)** to the canvas.
2. In the Properties view, click within the **sid** value field.
3. Type the **sid** value and press Enter to set the value.
4. Click within the **label** value field.
5. Type the label for the **List (Select1)** and then press Enter.  
Since you are not connecting the list to a data node set, you must turn off the **itemset** property.
6. Expand **XForms (select1)**, and **itemset**.
7. Turn off the **itemset** property.
8. Expand **XForms (select1)**, and **itemset**.
9. Click within the **XForms (Select1)**, **ref** value field.
10. Type the XPath of the node that you want to bind the list to.
11.  
You must now set the property that will store the user's choice from the list.
12. Expand **XForms (select1)**, **itemset**, and **value**.
13. Click within the **ref** value field and type the XPath of the node that will store the chosen list item value.  
Almost always, the **ref** value will be . (period).

Next, you add items to the list. Each item you add will be displayed to the user as a list item choice.

### Adding list items to a List (Select1):

You can add list items to a **List (Select1)**. Each item you add will be displayed to the user as a list item.

While this procedure uses **List (Select1)** as an example, you can also apply this procedure to the following XForms items:

- **List (Select)**
- **Popup (Select1)**
- **ComboBox (Select1)**

To add an item to a **List (Select1)**:

1. On the canvas, select the **List (Select1)** item that you want to add the list choices to.
2. In the Properties view, expand **XForms (select1)**.
3. Click within the **Item** value field. (This cell has <empty> as a value.)



- A drop down arrow button and a plus sign button appear.
4. Click the plus sign button.
    - An item is added to the list.



You must now set the list item's label value. This value will be what the user sees in the list.
  5. Expand the **Item,item**, and **label**.
  6. Click within the **Text** value field.
  7. Type the list item name and press Enter.
 

You must now set the item's **value** property. The value's **Text** property value defines what data is passed to the form's data layer.
  8. Expand **value**.
  9. Click within the **Text** value field.
  10. Type the value and then press Enter.
  11. Repeat steps 3 through 10 to add the remaining list items to the list.

### Reordering list items in a Select or Select1 list


List items appear in the list at run time in the order in which they were added to the list. You can reorder the list items that appear in a list.

To reorder list items:

1. On the canvas, click the XForms **List (Select)** or **List (Select1)** list.
2. In the Properties view, expand **XForms (select)** (or **XForms (select1)**), and **Item**.
3. Click  or  to move an item up or down in the list.

### Deleting list items from a Select or Select1 list

To delete a list item from a Select or Select1 list:

1. On the canvas, select the XForms **List (Select)** or **List (Select1)** list.
2. In the Properties view, expand **XForms (select)** (or **XForms (select1)**), and **Item**.
3. Click  to delete the list item.

### Setting the appearance of a Select (List)

This topic assumes that you have already added and configured a **Select (List)** XForms control to your form.

The list's **appearance** property determines how the **select** and **select1** option lists are displayed to the user. There are three possible settings for appearance:

- **full** — Expands the list so that the entire list is always visible.
- **compact** — Displays the list as a framed box list. Use with box lists only.
- **minimal** — Limits the list to one row in height unless it is being accessed by a user.

The **appearance** setting is dictated by the type of list you add to your form:

- checkgroups and radiogroups can only appear as **full**
- popups and comoboxes can only appear as **minimal**
- select1 lists can only appear as **compact**

The only list type appearance you can change is a **Select (List)**. You can set the appearance to **compact** or **minimal**.

To set the **List (Select)** option type's appearance:

1. On the canvas, click the **List (Select)** control.
2. In the Properties view, expand **XForms (select)**.
  - By default, the select option list's appearance is **compact**.
3. Click within the **appearance** value field.
4. Click the drop down arrow and select the either **compact** or **minimal**.

## XForms conditional items

Conditional items are items that are not displayed unless a pre defined condition is met. For example, consider a form that needs to be rendered in several languages. You could have three labels (English, French, and Spanish) for each field, or you could create the labels and fields as conditional items and allow the user to select their language of preference.

### Creating conditional items

Three XForms items are needed to create conditional items:

- **Switch** — Specifies what information is conditional. It also groups a set of **Case** XForms items. When you create a switch, a default pane and case is also created. A *pane* is an XFDL item that acts as a container for other items. You can see these objects in the Outline view.
- **Case** — Contains the different data options.
- **Trigger (Button)** — Lets you toggle between cases.

### Conditional items and the data model

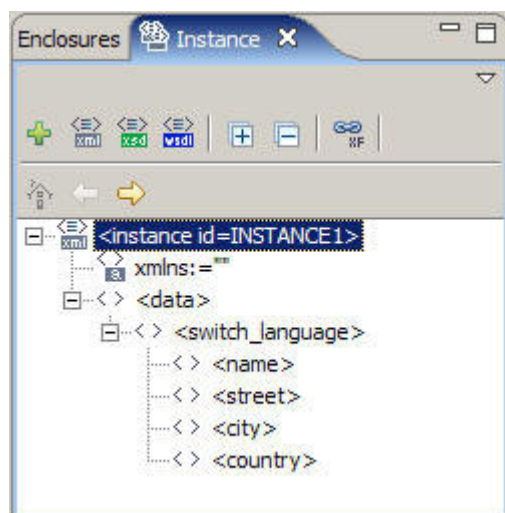
When you create a **Switch** item that contains conditional items, you must associate it with a parent data node.

Associating a **Switch** item with a parent node indicates that its child nodes have conditional associations. In other words, these nodes may be referenced by multiple conditional items in the user interface, but only retain the data provided by the active condition. The name of this node is entirely arbitrary, but it should be unique and reflect the purpose to which it will be used. You must also create the child nodes. You must make one child node for each item that will appear in the pane.

### Switch language example

In this example, conditional items provide text in different languages for fields that collect the user's name and address.

If you want a case to contain fields that will display and contain name, street, city, and country information, the `<switch_language>` node will need to have four child nodes; one for each piece of data:



## Adding an XForms Switch item

You use a **Pane (Switch)** to group one or more **Case** items.

To add a **Pane (Switch)** item and bind it to the form's data layer:

1. Add a **Pane (Switch)** to the canvas.
2. In the Instance view, click the data node set that you want to bind to the switch and drag it to the switch.  
The Ambiguous drop window is displayed.
3. Click **Button (Update)**.  
The switch is bound to the data node set.

Next, you can add more **Case** items to the **Pane (Switch)** or you can add items to the default **Case**.

## Adding Cases to a Switch

You can add one or more cases to a **Pane (Switch)**. Each **Case** contains a set of conditional user interface items that are displayed when the proper conditions are met.

You need to create a separate case for each set of conditional items.

**Note:** When you add a switch, a default case is added to your form.

To add a case to a switch:

1. Add a **Case** to the switch.  
  
**Note:** You will not be able to see the case in the **Design** editor. To select and work with a case, you must use the Outline view.
2. In the Properties view, expand **Identification**.
3. Click within the **id** value field.
4. Type the **id** for the case and then press Enter.
5. Repeat steps 1 through 4 to add as many cases to the switch as required.

Next, you must add the items that will display each case's conditional items.

## Adding items to a case

The items you add to a case represent the conditional items that will be displayed when the case is activated. For example, if you want to present three language choices to a user and you want to display and collect the user's name, street address, city and country, you would add four **Field (Input)** items to each of the three cases.

**Note:** You can add both XFDL and XForms items to a case.

To add items to a case:

1. In the Outline view, right-click the case that you want to add items to and click **Active case** → *The name of the case*.
2. Add the item to the case.
3. Position the item.
4. If the item you added is an XForms item, in the Instance view, drag the data node to the item that you want to bind.  
The item is bound to the data node.
5. Repeat step 2 for each XFDL item that you want to add to the case.
6. Repeat steps 2 through 4 for each XForms item that you want to add to the case. Using the switching languages example in "XForms conditional items" on page 148, you would add four **Field (Input)** items and bind each item to the appropriate child node.

Next, you must add and configure a toggle case button for each case so that when a user clicks the button, the correct items in the case are displayed. For detailed information on creating a toggle case button, see "Creating toggle case buttons."

## Setting the default case

The default case is the set of conditional items that appears in your form by default. For example, if you know that the majority of your form's users are Spanish speaking, you can set the case that contains the Spanish conditional items to be displayed as the default case.

To set the default case:

1. In the Outline view, click the case that you want to use as the default case.
2. In the Properties view, expand **XForms (case)**.
3. Set the **selected** property to **true**.
4. Make sure that the other cases have their **selected** property to **false**.

## Creating toggle case buttons

A toggle case button lets users select which case they want to use. Clicking a toggle button displays the case associated with the toggle button.

Each case requires a toggle case button. For example, if you want to present three language choices to a user you would add three toggle buttons.

**Important:** If you are going to add a toggle case button to a case, thoroughly test your form's logic to ensure that the correct buttons are always visible to the user.

To create a toggle case button:

1. Add a **Button (Trigger)** to the form.
2. Position the button where you want it to appear. Make sure that the button is not in the case, otherwise, it will not be visible to users.
3. In the Properties view, expand **XForms (trigger)**, and **label**.
4. Click within the **Text** value field.
5. In the **Text** property, type the button name that you want the user to see and then press Enter. Using the language example, you could type the language name.
6. Repeat steps 1 through 5 to create a toggle button for each case you want to conditionally display.





Next, you must change the button to a toggle case button and then link it to a case so that when the user clicks it, the button displays the case's items.

#### **Adding a toggle action to a toggle case button:**

This procedure assumes that you have already added a **Button (Trigger)** to your form.

You must change the button to a toggle case button and then link it to a case so that when the user clicks it, the button displays the case's items.

To add a toggle action to toggle between cases:

1. On the canvas, click the **Button (Trigger)** you want to change into a toggle button.
2. Expand **XForms (trigger)**, **Actions**, and **action**.
3. Click within the **Actions** value field.  
The drop down arrow  and Add  buttons are displayed.
4. Click the drop down arrow button  and select **toggle**.
5. Click the add button .
6. Expand **Actions**, and **toggle**.
7. Click within the **case** value field.
8. Type the name of the case you want to display when a user clicks the toggle action button.
9. Repeat steps 1 through 8 for each **Button (Trigger)** you want to add to the form.

**Note:** You should have one toggle case button for each case on the page.

## **XForms tables**

The following methods are available to create XForms tables:

- **Wizard** — by using **Table (Repeat) by Wizard**.
- **Manual** — by using **Table (Repeat)**.

### **Creating XForms tables using a wizard**

An XForms table lets you arrange data into rows of items, making data easier to interpret and forms easier to complete.

The following table wizard methods are available:

- **Simple Setup** — Enables you to create a complete table. All the necessary background work is done for you. An XForms data instance is created for you to collect the data from the form. You simply specify the number of columns and rows to display in the table.
- **Advanced Setup (using existing data)** — Enables you to use an existing instance in the form or another instance on your computer or shared network directory (xml file format). You specify the number of columns and rows to display in the table.

The wizard allows you to choose general display and configuration settings such as “+” and “-” buttons for the user to add or delete rows, table lines, borders, and row formatting.


For detailed information on how to use the table wizard, see “Creating a simple XForms table using the wizard” and “Creating an advanced XForms table using the wizard” on page 153.

**Note:** Do not insert a table wizard object inside another table wizard object. References to the “+” and “-” buttons will not work correctly. To create this table layout, see “Creating XForms tables manually” on page 153.

#### Creating a simple XForms table using the wizard:

By choosing **Simple Setup**, you can create a table that allows the user to add or delete rows of information.

To complete a simple XForms table setup:

1. In the Palette, click **Table (Repeat) by Wizard**.
2. Click on the canvas where you want the table to be positioned.  
The Table Wizard opens with **Simple Setup** selected.
3. Click **Next**.
4. Within the **New Element Name** field, type the column name you want to display in your table.
5. Click  to add to the list.
6. Repeat steps 4 through 5 for each element name used to control the column data in the table.

To remove an element name from the list, select the element and click .

To change the order of the element names, select the element and click the **up arrow** button or the **down arrow** button.

7. In the **Enter Number of Initial Rows** field, type a number.
8. If you want to create a new instance or use an existing instance, click **Advanced** and set the options.
9. Click **Next**.
10. In the **Display Columns** list, select a column name listed.
11. Set any of the following **Details** for each column name:
  - **Display as** — Set the display to either **Text (Read/Write)** or **Text (Read)**.
  - **Include Header** — Select or clear the check box to display or hide the column header name.
  - **Header** — Type the column name you want displayed on the table.
  - **Width (Pixels)** — Set the width of the selected column. The default is 100 pixels.

- **Show Border** — Turn on or off the borders. Default is on.
12. Click **Next**.
  13. Set any of the display and configuration settings.
  14. Click **Finish**.

### Creating an advanced XForms table using the wizard:

By choosing the **Advanced Setup (Using Existing Data)** you can use an existing instance in the form or use another instance (xml file format) on your computer or shared network directory.

To complete the Advanced XForms table setup:

1. In the Palette, click **Table (Repeat) by Wizard**.
2. Click on the canvas.  
The Table Wizard Setup opens.
3. Click **Advanced Setup (Using Existing data)**.
4. Click **Next**.
5. Select the instance you want to use to control the data in the table:
  - To use an instance already being used in your form, in the Instance column select the instance
  - To use an instance saved on your computer or network, click **Add Instance** and select the file. Then in the **Instance** column, select the instance.
6. In the **Instance Data** column, select the instance data to use.  
An XForms data instance defines the XML template for the data that will be collected from the form.
7. Click **Next**.
8. Set the configure details.
  - **Available Columns** — The available child elements for the instance data selected. You can reuse the elements. By adding them to your table you are adding columns to your table.
  - **Display Columns** — The column titles displayed.
  - **Details** — The following options are available:
    - **Display as** — Set the display to either **Text (Read/Write)** or **Text (Read)**.
    - **Include Header** — Select or clear the check box to display or hide the column header name.
    - **Header** — Type the column name you want displayed on the table.
    - **Width (Pixels)** — Set the width of the selected column. The default is 100 pixels.
    - **Show Border** — Turn on or off the borders. The Default is on.
9. Click **Next**.
10. Set any of the display and configuration settings.
11. Click **Finish**.

### Creating XForms tables manually

To create an XForms table manually in the user interface, you use the repeat item. You then add XFDL and XForms items to the repeat item to create a template row. You can then create an Insert Row button and a Delete Row button to add or remove rows from the table.

## Binding XForms tables to the data model

You must bind the repeat item to a node set (parent node) in your data model. You must also bind each XForms item in the template row to the child nodes of the node set you bound to the repeat item.

### XForms table example

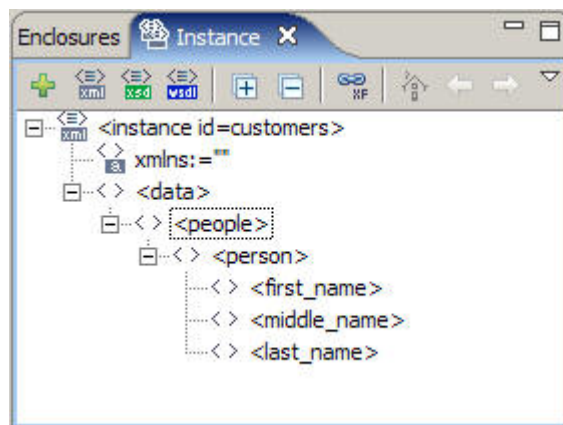
The topics that comprise this section will use the following scenario to develop an XForms table.

#### Scenario:

You want to collect and display the first, middle and last names of customers.

#### Table node set:

In this example, the following data instance will be used:



**Note:** You may want to create this data instance so that you can develop the example XForms table. For detailed information about creating data instances, see "Creating an XForms data instance" on page 132.

#### Table User Interface:

To mirror the node set, we will make a table consisting of three XForms **Field (Input)** items that will contain and display the person's first, middle, and last names in one table row.

First name	Middle name	Last name
<input type="text"/>	<input type="text"/>	<input type="text"/>

**Note:** The steps to create the template row are discussed in "Adding items to an XForms table" on page 155.

#### Binding an XForms table:

You must bind the XForms table to the appropriate node set in the data instance.



To bind an XForms table:

1. In the Instance view, click the node set that you want to bind to the XForms table and drag it to the repeat item.  
The Ambiguous drop window is displayed.
2. Click **Update**.  
The repeat is bound to the node set. Using the example, the nodeset value is **instance('customers')/people/person**.
3. If you want to set which table row gets focus when the form is opened, set the **startindex** property.  
The index starts with 1, rather than 0. If **startindex** is not used, the focus defaults to the first row in the table.

Next, you must add items to the repeat item.

### Adding items to an XForms table:

You can add XFDL and XForms items to an XForms **Repeat**. The items you add to the **Repeat** form the table's template, or repeating row. Each time a user clicks the Insert Row button, all the items you added to the template row will be repeated in a new row.

To add items to a table:

1. Add the item that you want to add to the **Table (Repeat)** item.
2. Position the item where you want it to be displayed in the template row. For detailed information about positioning items, see "Alignment types" on page 44.
3. If you want to add a label to the item:
  - a. In the Properties view, expand **General**.
  - b. Click within the **label** value field.
  - c. Type the label text for the item and press Enter.
4. If you added an XForms item:
  - a. In the Instance view, drag the appropriate data node to the item. The item is bound to the node.
5. Repeat steps 1 through 4 to add all of the items that you want to include in the template row. Using the section example, you would add three XForms **Field (Input)** items to the repeat item. Bind the first input field to <first\_name>. Bind the second input field to <middle\_name>. Bind the third input field to <last\_name>.

Following the example, your form should resemble the following:

First name	Middle name	Last name
<input type="text"/>	<input type="text"/>	<input type="text"/>

Next, you can add an Insert Row button.

### Adding an Insert Row button:

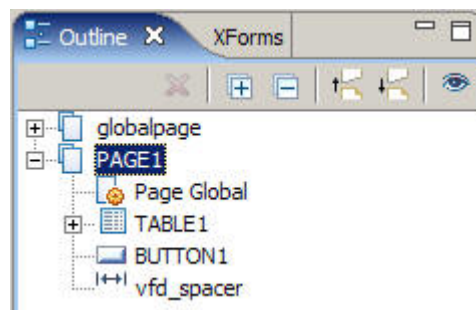
An Insert table row button inserts a template row at the bottom of an XForms table.

To add and position an Insert Row button:

1. Add a **Button (Trigger)** to the canvas.
2. Use relative positioning to position the button **below** the table. For detailed information about relative positioning, see “Aligning items” on page 43.

The button repositions itself immediately under the table.

**Note:** If the button does not reposition itself immediately below the table, there may be an issue with your build order. If the button does not come after the table in the build order, then the button cannot refer to the table. In the Outline view, make sure that the button is listed immediately beneath (and not inside) the table.



For detailed information about build order, see “Changing the build order of items” on page 42.

3. In the Properties view, expand **XForms (trigger)** and **label**.
4. Click within the **Text** value field.
5. Type the label for the button and press Enter.

Next, you must configure the **Button (Trigger)** so that when it is clicked by the user, a template row is inserted into the table.

#### Configuring an Insert Row button:

The Insert Row button is a configured XForms **Button (Trigger)** item that adds a template row to a table.

In order for the Insert Row button to create additional table rows when it is clicked by the user, it must contain two XForms actions:


- **insert** — The **insert** action adds the new node by cloning the final node in a node set.
- **setindex** — The **setindex** action specifies the position of the focus in the index. The index is the method used by each repeat item to keep track of which item of the form’s repeat item currently has the focus.

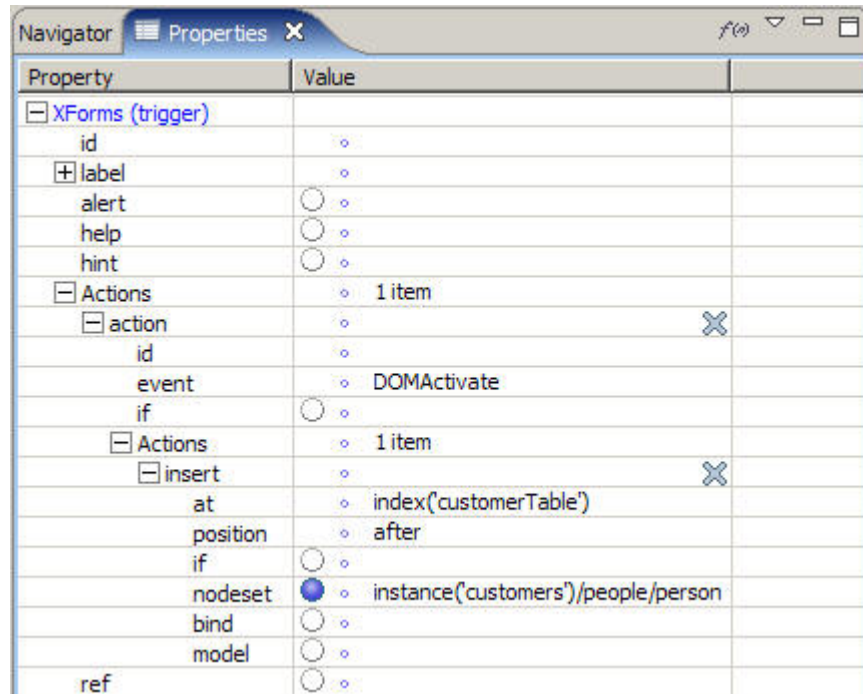
#### Configuring the Insert Row button insert action:



To configure the Insert Row button **insert** action:

1. On the canvas, click the button that you want to use to insert a row.
2. In the Properties view, expand **XForms (trigger)**, **Actions**, and **action**.
3. Click within the **Actions** value field.

The drop down arrow  and add  buttons are displayed.

4. Click the drop down arrow button  and select **insert**.
  5. Expand **Actions**, and **insert**.
  6. Set **at** to `index('table XForms id value')`
  7. Set **position** to **after**.
  8. Set **nodeset** to the path of the node set you want to clone.
- The **insert** action's properties should resemble the following.




Property	Value
<input type="checkbox"/> XForms (trigger)	
id	<input type="radio"/>
<input checked="" type="checkbox"/> label	<input type="radio"/>
alert	<input type="radio"/>
help	<input type="radio"/>
hint	<input type="radio"/>
<input type="checkbox"/> Actions	<input type="radio"/> 1 item
<input type="checkbox"/> action	<input type="radio"/> 
id	<input type="radio"/>
event	<input type="radio"/> DOMActivate
if	<input type="radio"/>
<input type="checkbox"/> Actions	<input type="radio"/> 1 item
<input type="checkbox"/> insert	<input type="radio"/> 
at	<input type="radio"/> index('customerTable')
position	<input type="radio"/> after
if	<input type="radio"/>
nodeset	<input checked="" type="radio"/> instance('customers')/people/person
bind	<input type="radio"/>
model	<input type="radio"/>
ref	<input type="radio"/>



Next, you must add and configure the **setindex** action.

#### Configuring the Insert Row button **setindex** action:

To configure the Insert Row button **setindex** action:

1. In the Properties view, expand **XForms (trigger)**, **Actions**, and **action**.
2. Click within the **Actions** value field.

The drop down arrow  and add  buttons are displayed.

3. Click the drop down arrow button  and select **setindex**.
4. Click the add button .
5. Expand **setindex**.
6. Set **index** to `index('table XForms id value') - if(index('table XForms id value')=count(parent_node/child_node), 1, 0)`
7. Set **repeat** to the path to the node set you want to repeat.

The Insert Row button's actions should resemble the following.

[-] Actions	o 1 item	
[-] action	o	✕
[-] Actions	o 2 items	
[-] insert	o	↓ ✕
at	o index('customerTable')	
position	o after	
if	<input type="radio"/> o	
nodeset	<input checked="" type="radio"/> o people/person	
bind	<input type="radio"/> o	
model	<input type="radio"/> o	
[-] setindex	o	↑ ✕
index	o index('customerTable') - if(index('customerTable')=count(people/person), 1, 0)	
repeat	o instance('customers')/people	
if	<input type="radio"/> o	
id	o	
event	o DOMActivate	
if	<input type="radio"/> o	

Now, you can add a Delete Row button.

### Creating a Delete Row button:

You must first add an Insert Row button to the form. For detailed information on adding an Insert Row button, see “Adding an Insert Row button” on page 155.

A Delete Row button removes the XForms table row that has focus.

To add and position a Delete Row button:



1. Add a **Button (Trigger)** to the form.
2. Use relative positioning to position the button after the **Insert Row** button. For detailed information about relative positioning, see “Aligning items” on page 43.
3. In the Properties view, expand **XForms (trigger)**, and **label**.
4. Click within the **Text** value field.
5. Type **Delete Row** and press Enter.


Next you must add the **delete** action to the button.

### Configuring the Delete Row button delete action:

To configure the Delete Row button delete action:

1. On the canvas, click the button that you want to use to delete a row.
2. In the **XForms (trigger)** properties, expand **Actions** and **action**.
3. Click within the **Actions** value field.

The drop down arrow  and add  buttons are displayed.

4. Click the drop down arrow button  and select **delete**. Next, you must set the **delete** action’s properties.
5. Expand **Actions** and then **delete**.
6. Set **at** to `index('table_id_name')`
7. Set **nodeset** to `instance('instance_name')/.../node set)`

The Delete Row button’s delete action should resemble the following.

Property	Value
[-] XForms (trigger)	
id	o
[-] label	o
[+] output	o
Text	o Delete row
ref	o
bind	o
model	o
alert	o
help	o
hint	o
[-] Actions	o 1 item
[-] action	o
id	o
event	o DOMActivate
if	o
[-] Actions	o 1 item
[-] delete	o
at	o index('customerTable')
if	o
nodeset	o instance('customers')/people/person
bind	o
model	o
ref	o
bind	o
model	o
[-] General	

## XForms help messages

XForms help messages let you use dynamic data from your instance to create additional help or warning messages for your forms.

If you want to add help and accessibility messages to your XForms items, you have three options:

- Alert
- Help
- Hint

You can also use the following XFDL help messages to provide assistance to your users:

- Help
- Acclabel
- Message (in the **Format**, **format**, **constraints** property group)

For detailed information about XFDL help messages, see “Help messages” on page 54.

### Adding an XForms alert message

An XForms **alert** message is displayed to the user if they enter invalid information. This is equivalent to the **message** property in the XForms item’s **Format**, **format**, **constraints** property. If both an alert and a message are provided for the XForms item, then the XFDL message overrides the alert.

To add an XForms alert message:

1. On the canvas, select the XForms item to which you want to add an alert message.
2. In the Properties view, expand **XForms** (*option type*).
3. Click the **Alert** property's On button.
4. Expand **alert**.
5. Click within the **Text** value field.
6. Type the alert message and press Enter.

### **Adding an XForms help message**

Help provides a message that is displayed to the user if they enter help mode. This message is generally longer than a hint message, and is intended to provide detailed help to the user. Although there is no direct equivalent in XFDL, XForms help is treated like the help option, and is displayed as a tooltip when the user enters help mode. If an item contains both an XForms hint and an XForms help message, then the help is appended to the hint. Furthermore, if an item contains both an XForms help and an XFDL help option, then the XFDL help option overrides the XForms help.

To add an XForms help message:

1. In the canvas, select the XForms item to which you want to add a help message.
2. In the Properties view, expand **XForms** (*option type*).
3. Click the **Help** property's On button.
4. Expand **help**.
5. Click within the **Text** value field.
6. Type the help message and press Enter.

### **Adding an XForms hint message**

A hint message provides a message that is displayed to the user if they enter help mode. This message is generally a short instruction, such as telling the user what format is valid for a specific field, and is displayed as a tooltip. This is equivalent to the help option. If an item contains both an XForms hint and an XFDL help option, then the XFDL help option overrides the XForms hint.

To add an XForms hint message:

1. In the canvas, select the XForms item to which you want to add a hint message.
2. In the Properties view, expand **XForms** (*option type*).
3. Click the **Hint** property's On button.
4. Expand **hint**.
5. Click within the **Text** value field.
6. Type the hint message and then press Enter.

## **Converting XFDL items to XForms items**

You can convert an existing XFDL item to an XForms item, allowing you to bind it to the data layer. If you convert an existing XFDL form to XForms, you must convert the items individually.

To convert an XFDL item:

On the canvas, right-click an existing XFDL item and select **Convert Item***XForms item* .

To see the converted item's XForms properties:

1. Select the item you just converted.
2. In the Properties view, expand **XForms (item name)**.

Remember to bind the converted item to a data node.

---

## XForms binding

*Binding* is a link between the data layer and the form's user interface (UI) layer.

Binding lets you:

- Synchronize the data model with the form presentation layer; if the value of one changes, the other linked elements are updated to reflect the changes.
- Place constraints, calculations, validations and limitations on what data the user enters.

You can bind UI controls to the data instance using one of two methods:

- **ref** or **nodeset** — Creates a direct link between a UI element and a data element in the XForms instance using an XPath reference.
- **bind** — Creates an indirect link between a UI element and a data element in the XForms data instance using a model bind.

### Binding using ref or nodeset

With the exception of the XForms **repeat** items, each XForms item has a **ref** property that you can set to bind the XForms item directly to a single node in a data instance using the node's XPath reference.

XForms **repeat**, **select** and **select1** items have a **nodeset** property. Unlike **ref** — which limits the bind to a single node — the **nodeset** property lets you bind a **repeat**, **select** or **select1** items to a set of nodes using the node set's XPath reference.

### Binding using the ref property

When you use the XForms item's **ref** property to create a bind, you create a direct link between an XForms item, such as an **Field (Input)**, and a data node in the data instance using an XPath reference.

**Note:** Binding a data node to an XFDL item automatically converts it to an XForms item.

To bind an item to a data node using the **ref** property:

1. In the Instance view, click the node that you want to bind.
2. With your mouse, drag the node to the item that you want to bind.  
The XPath bind is displayed in the item's **ref** property.

To see the **ref** property value:

1. Select the item on the canvas.

2. In the **Properties** view, expand **XForms (option type)**  
The bind appears in the **ref** property value.

## Binding using the nodeset property

The following XForms items have **nodeset** properties:

- **Select** list types
- **Select1** list types
- **Repeat**.

These items are containers that hold other XForms items. Similarly, in the data layer a *node set* (or parent node) is a container for child nodes.

To create a relationship between the presentation and data layers you bind the XForms container item to the node set using the **nodeset** property. Then you bind the XForms items inside the container to the child nodes inside the parent node using the **ref** property.

For detailed information on how to bind an XForms **Select** item list using the **nodeset** property, see “Displaying list items from a data node set” on page 142.

For detailed information on how to bind an XForms **Repeat** item using the **nodeset** property, see “Binding using ref or nodeset” on page 161.

## Binding using bind

Each XForms item has a **bind** property that lets you bind the item to a model bind. The **bind** property creates an indirect link between an XForms item and a data node in the XForms data instance.

### Binding using the bind property

To bind an XForms item to a model bind, you must have an existing model bind. For detailed information on model binds, see “Creating model binds” on page 164.

When you use the XForms item’s **bind** property to create a bind, you create an indirect bind between an XForms item and a data node in the XForms data instance using a model bind. This binding is then automatically extended to the data element that the bind affects.

**Note:** You cannot bind to nested binds. You can only bind to the outermost bind in any nested structure.

To bind an XForms item to the data instance using the **bind** property:

1. On the canvas, select the XForms item that you want to bind.
2. In the Properties view, expand **XForms (option type)**.
3. In the **bind** property value, select the bind id from the list.
  - The bind is displayed in the XForms item’s **bind** property.

## XForms model binds

Once you have created a data instance, you can set the properties of its nodes. This is done using a model bind.

You use a *model bind* to perform special calculations or place limitations on user data. For example, you might want to perform a calculation on a certain node or ensure that supplying certain data is mandatory.



Each XForms model can have one or more associated model binds.

## Model bind properties

Every model bind contains one or more model item properties. These properties describe the way the model bind modifies its associated node. These modifications include determining the value of nodes, their validity, or relevancy.

Model bind properties let you:

- Name the model bind (optional)
- Determine the node or nodeset that is affected by the model bind (required)
- Describe the way the model bind effects the element (required)

**id** Lets you give a globally unique name to your model bind. This property is optional, but necessary if you want to refer to the model bind elsewhere in your form.

### nodeset

Identifies which element is affected by the model bind, it must refer to a node in a data instance. Every model bind is associated with a nodeset, either directly, or in the case of nested binds, through inheritance from a parent bind.

## Model Item Properties

Every model bind contains one or more model item properties. These properties describe the way the model bind modifies its associated node. These modifications include determining the value of nodes, their validity, or relevancy and are the following:

### Calculate

The *calculate* property defines a calculation that determines the value of the associated node. It lets you add mathematical formulas and computed logic to your data instance. The essential elements of the calculation are XPath expressions combined with normal mathematical expressions.

The essential elements of the calculation are XPath expressions combined with normal mathematical expressions.

### Constraint

The *constraint* property determines whether an associated node is valid.

For example, if you wanted to ensure that a field contains a number higher than 0 but less than 10, you would use **constraint** to prevent the form from accepting a value that was outside of that range.

You can use both relational and logical operators. Relational operators are character sets that describe how one thing relates to another. For example, the greater than and less than signs are relational operators. Logical operators let you create more complex computes with logical or and logical and.

### Readonly

The *readonly* property determines whether the data in the associated node can be changed.

**Readonly** accepts any XPath expression as its setting, but the result is always converted to either true() or false().

The default value of `readonly` is `false()`, as most nodes will accept input from the user. However, if the model bind includes a `calculate` property, the node automatically has a `readonly` of `true()`, as the value of the node will be based on a calculation, and not directly on input from the user. Furthermore, if a node is set to `readonly`, then all of its child nodes automatically inherit the `readonly` setting.

#### Relevant

The `relevant` property determines whether a node is displayed to the user or included in XForms submissions.

#### Required

The `required` property determines whether a node requires mandatory user input.

**Type** The `type` property sets the data node to be a particular data type.

## Creating model binds

To create a model bind:

1. In the XForms view, select the model that you want to add the model bind to.
2. Right-click and select **Create Bind**.
3. In the Properties view, expand **Identification**.
4. Type a descriptive, unique bind name in the `id` value.

The `id` property is optional, but necessary if you want to refer to the bind elsewhere in your form.

**Note:** IDs are only allowed on parent binds. They are not supported on binds nested inside another bind.

5. Expand the **General** property node.
6. In the `nodeset` property value, type the path location of the node set whose elements you want to bind.

The `nodeset` value identifies which element is affected by the model bind and must refer to a node in a data instance. Every model bind is associated with a node set, either directly, or in the case of nested binds, through inheritance from a parent bind.

If necessary, you can create multiple model binds for a single node set.

Next, you must define the model item properties. For detailed information on model item properties, see “Model bind properties” on page 163.

## Highlighting bound XForms items

You can analyze the XForms items you have in the **Design** editor to see if they have been bound. You can do this from the **View** menu:

- **View** → **Highlight Items with XForms Binds** — Use this option to highlight XForms items that are associated with XML instance elements.
- **View** → **Highlight Missing XForms Binds** — Use this option to highlight XForms items that do not reference XML instance elements.
- **View** → **Turn off Highlight Mode** — Use this option to turn off the highlighting for XForms items with/without references. These options rotate through a three-way state change in the **View** menu.

**Note:** The highlight color can be customized in the Designer’s preferences.

---

## XForms submissions

A *submission* is a set of rules that defines what form data is submitted, how the data is submitted, and where the data goes.

When submitting data from a form that contains an XForms model, you can submit a particular data instance. Submitting a data instance 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.

In addition to defining the submission's rules, you must also create a submission button that is linked to the submission.

If your form has more than one model, you can create a set of submission rules for each model. Additionally, you can create a set of submission rules for each data instance.

### Adding submissions to an XForms model

You can add one or more submissions to an XForms model. The number of submissions you add to a form will depend on how your form collects data from the presentation layer to the data layer and how many different submissions you want the user to make.

To add a submission to an XForms model:

1. In the XForms view, select the model that you want to add the submission to.
2. Right-click and select **Create Submission**.
  - The Designer creates a submission.

### Adding a submission to an XForms data instance

To add a submission to an XForms instance:

1. Select the XForms view.
2. Select the instance that you want to add the submission to.
3. Right-click and select **Create Submission**.
  - The Designer adds a submission to the XForms model and the data instance id appears in the submission's XForms **instance** property value.

### Naming submissions

To name a submission:

1. In the XForms view, select the submission. You might need to expand the model to see the submission.
2. In the Properties view, expand **Identification** and then name the submission by entering an **id** value.

### Setting which data is submitted

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 to extract the data instance.

By default, the first data instance in an XForms form is submitted. If you want to submit something other than the entire first data instance, you must provide one of the following:

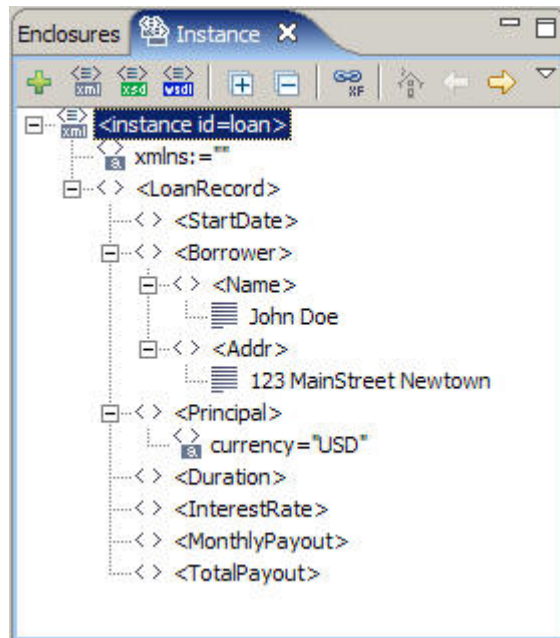
- An XPath expression that specifies a node (and its children) for submission.
- The name of a bind that references the node set you want to submit.

## Specifying nodes

You can choose to submit an entire data instance or only a portion of the instance. This is done using the submission's `ref` property.

When submitting only a portion of a data instance, you must identify the root element of the submission. The root element determines which portion of the instance is submitted, since only the root element and its children are sent.

For example, consider the following data instance.



If you wanted to submit the entire instance, you would need to select the root node of the instance. The first tag in an instance is its root node.

If you wanted to select only a portion of the instance, you would have to select the root node of the data portion you wanted to submit. For example, if you only wanted to submit the `<Borrower>` information, you would reference the `<Borrower>` node.

Specifying a parent node includes all of its child nodes in the submission. In the following example, the `ref` property indicates that the submission data should consist of the `<Borrower>` node and its child nodes, `<Name>` and `<Address>`:

```
instance('loan')/Borrower
```

The submission's `ref` property also lets you use an XPath expression that computes the node that will be submitted. The default value of `ref` is `/` (forward slash).

## Specifying a bind

Another way of stipulating which data you want to submit is by referencing a model bind. This is done using the submission's **bind** property.

By referencing a model bind, you specify the bind's node set as the data you want to submit. Furthermore, all of the limitations you placed on the bind are relevant to the submission data. For example, if the bind indicates that certain of its nodes are not valid, then those nodes would not be included in the submission, even though they were part of the indicated node set.

You can specify a particular bind by referring to its **id** attribute. For example, if the bind's **id** property is **loan\_history**, then the submission's **bind** property should be **loan\_history**.

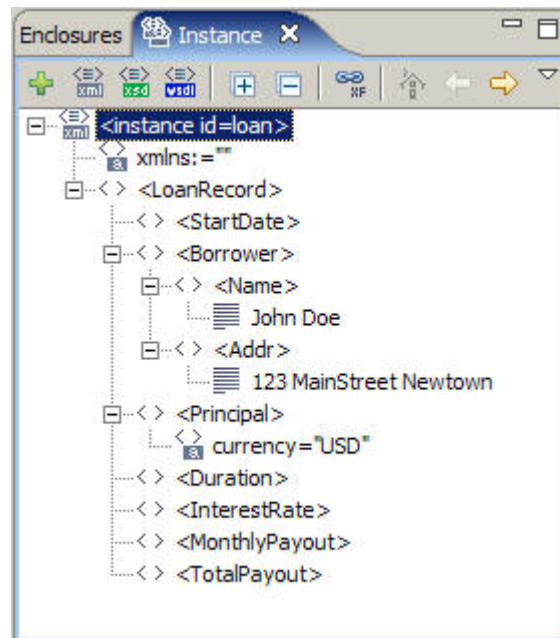
## Setting which data is submitted using ref

To set the data to be submitted using **ref**:

1. In the XForms view, select the submission. You might need to expand the model to see the submission.
2. In the Properties view, expand **XForms** and then click the **ref** property's **On** button.
3. Enter the reference.

**Tip:** In the Instance view, select the node that you want to use as the reference, right-click and select **Copy Reference**. Paste the reference in the **ref** property.

Using the following instance as an example.



Borrower is a child node of LoanRecord. If you want to submit the Borrower node, the **ref** value would be: **instance('loan')/Borrower**

## Setting which data is submitted using bind

To set the data to be submitted using **bind**:

1. In the XForms view, select the submission. You might need to expand the model to see the submission.

2. In the Properties view, expand **XForms**.
3. Click the **bind** property's **On** button.
4. Enter the bind **id**. Remember, the submission **id** is the name of the submission. The bind **id** is the name of the bind that contains the node set you want to submit.

## Setting the submission type

You must also set each submission's **method** property. This property describes how the submission will be performed. The **method** property can have either of three values:

- **post** — Serializes the data and sends it as XML.
- **get** — Serializes the data and sends it as URL encoded data.
- **put** — Serializes the data and saves it as a file instead of submitting it.

To set the submission type:

1. In the XForms view, select the submission. You might need to expand the model to see the submission.
  - The submission's properties are displayed in the Properties view.
2. In the Properties view, expand **XForms** and then from the method list, select the submission method.

## Setting the target URL for a submission

You set the submission's target URL in the submission's **action** property.

**Note:** The Designer supports both `http:` and `https:` protocols. If you are using the `put` method and want to save your submission as a file, you must use the **file:** scheme. This indicates the directory and file in which the submission will be saved. For example: **file:C:\Documents and Settings\curchen\xforms\PO\instance.xml**

There are some limitations on where you can save submission data. Put submissions will fail if you try to save them to the following locations:

- The Program Files directory.
- The system drive, the Windows directory, or the Windows System directory.
- Temporary directories.
- A directory outside the folder sub tree containing the originating file.

To set the target URL for a submission:

1. In the XForms view, select the submission. You might need to expand the model to see the submission.
2. In the Properties view, expand **XForms** and then click the **action** property's **On** button.
3. Type the full path of the target URL (including the **http://** prefix).

**Note:** You can only list one URL in the action attribute.

## Creating a Submission button

A submission button lets the user submit the instance data to a processing server. The button triggers the submission, and the data submitted is determined by combining the filters for the button with the submission rules in the data model.

To create a submission button:

1. In the Palette, expand **Standard Library**.
2. Click **Button (Submit)**.
3. Add the button to the page.
4. In the Properties view, expand **XForms**.
5. Expand **submit**.
6. Click the **submission** property's **On** button and from the drop down list, select the submission name.

For detailed information about XForms buttons, see the *Workplace Forms XFDL Specification* document.

## XForms Smartfill

*Smartfill* automatically fills out portions of a form in the Viewer. This is accomplished by storing commonly used information, such as the user's name and address, on the user's computer. The Viewer can then access this information at any time, using it to automatically complete sections of forms that require it.

The first time the Viewer opens a Smartfill form, the user submits instance data from the form which is then saved as an XML file on the user's computer. Thereafter, each time the Viewer encounters a form — be it the same form or a new form — that uses the same instance, the Viewer retrieves the saved XML file and automatically completes that section of the form for the user.

By default, when data is returned by the submission, it replaces the submitted form. For example, when a user submits a form, the server often returns a reply that indicates that the submission was received. Generally, this notice entirely replaces the submitted form.

Instead of using Smartfill to simply replace the entire form, you can replace a single instance or even ignore the returned data. This is done using the submission's **replace** property. This property indicates whether the returned data should replace the entire form, a data instance, or be ignored.

Replace has three possible options:

- **all** — The returned data replaces the entire form.
- **instance** — The returned data only replaces the submitted instance.
- **none** — The returned data is ignored.

## Using XForms Smartfill

In order to use Smartfill, you need the following:

In the form that will collect and submit Smartfill data:

- XForms items to store the data inputted by the user.
- A data instance to store the Smartfill data.
- A submission to collect user data and then store it on the user's local machine.



- A Submit button to trigger the submission of the user data to be stored.

In the form that will display Smartfill data:

- XForms items to display the data previously inputted by the user.
- A data instance identical to the form that submitted the Smartfill data.
- A submission to retrieve and display the data stored on the user's local machine.

## Collecting and submitting Smartfill data

Smartfill automatically fills out portions of a form in the Viewer by retrieving previously submitted data stored on a user's local machine. Prior to using Smartfill to display data, you need to create a form that collects and submits user inputted data.

To collect and submit Smartfill data:

1. In the Instance view, design the data instance that will collect the Smartfill data.
2. Design the form's presentation layer by adding the XForms items that will collect the Smartfill data.
3. Bind the XForms items to the appropriate data nodes.
4. In the XForms view, create a submission.
5. In the Properties view, define the following submission properties:
  - a. **Identification** -> **id** = unique submission name
  - b. **XForms** -> **ref** = the name of the instance you created in step 1
  - c. **XForms** -> **action** = file:\\\\[LocalHost] For example: file:\\\\C:\temp\name.xml
  - d. **XForms** -> **method** = put
6. Add a **Trigger (Submit)** button to your form.
7. In the Properties view, define the following submission button properties:
  - a. **XForms** -> **submission** = the id of the submission you named in step 5, a.
  - b. **General** -> **sid** = the button name

Next, you must create a form that will display the Smartfill data.

### Related concepts

"XForms data instances" on page 131

"XForms items" on page 138

"XForms binding" on page 161

"XForms submissions" on page 165

## Displaying Smartfill data

Once you have designed and configured the form that collects and submits the Smartfill data, you can create the form that will display the Smartfill data.

To display Smartfill data:

1. In the Instance view, design the data instance that will display the Smartfill data.
 

**Note:** This data instance must be identical to the instance that collects the Smartfill data.
2. Design the form's presentation layer by adding the XForms items that will display the Smartfill data.



3. Bind the XForms items to the appropriate data nodes.
4. In the XForms view, create a submission.
5. In the Properties view, expand **Identification** → **id**.
6. Type a unique submission name.
7. Expand **XForms** and in the **ref** property enter the name of the instance you created in step 1.
8. In the **action** property, type the value of the submitted file.
  - The **action** value has to be identical to that of the submission that submitted the Smartfill data. For example file: \\C:\temp\name.xml
9. In the **method** property, select **get**.
10. In the **replace** property, select **replace**.
11. In the XForms view, right-click the model and select **Actions** → **Create send**. If your form contains more than one model, select the model that contains the instance you designed in step 1.
  - A send action is added to the model.
12. In the Properties view, expand **XForms**.
13. In the **send** action's **submission** property, type the id of the submission you named in step 6.
14. In the **send** action's **event** property, select **xforms-ready**.



---

## XML Model

**Note:** If you are creating a new form and want to add a data layer to it, you may wish to consider using the W3C XForms 1.0 standard, supported by Workplace Forms 2.6 and newer. For detailed information on XForms, see “XForms” on page 129.

The XML Model separates the form’s data layer from the presentation (user interface) layer. This is very useful because rather than having to parse an entire XFDL form and then extract the user data you want to collect from the form, you can use the XML Model to define what data to pre-populate in the form and what user data to collect from the form.

Furthermore, you can bind individual nodes in the XML Model to one or more XFDL items. This binding causes the layers to share data. If the value in one layer changes, the other layer is updated to mirror that change.

Once submitted, the XML block that results from the XML Model can be easily integrated with other XML processors.

To effectively use the XML Model you should have a thorough understanding of both XML and your data model. As well, this document does not explain everything about the XML Model. For detailed information about the XML Model, see the *Workplace Forms Using the XML Data Model* and the *Workplace Forms XFDL Specification* documents.

**Note:** You cannot have both an XML Model and an XForms model in the same form; they are mutually exclusive.

### XML Model components

The XML Model contains three core components:

- **Data Instances** — Data instances are arbitrary blocks of XML. An XML Model may contain any number of data instances, and each instance is created to serve a particular purpose. Additionally, each data instance can be linked to a button on the form that will trigger the submission of that instance, stripping away the rest of the form description. For detailed information about designing XML data instances, see “Designing data instances” on page 175.
- **Bindings** — Each data instance has associated bindings. Bindings are used to bind a data element to form items, ensuring that the two are synchronized; if the value of one element changes, the other bound elements are updated to reflect the changes. For detailed information about XML model binding, see “XML Model binding” on page 177.
- **Submissions** — Each data instance may have an associated set of submission rules. These rules control how a data instance is transmitted when it is submitted for processing. This is an optional feature, and is only necessary when you want to submit the data instance by itself, without the rest of the form. There are many cases in which you may want to submit the entire form, and then retrieve the data instance from the form during processing. This is particularly true when you are using signatures on your forms. For detailed information about XML model submissions, see “XML Model submissions” on page 178.

---

## Displaying XML Model views

You use the XML Model and XML Model Instance views to create a form using XML model. However, the Designer's default perspective does not display these views.

To display the XML Model and XML Model Instance views:

1. Close the XForms and Instance views.
2. Click **Window** → **Show View** → **XML Model**.
3. Repeat steps 1 through 2, only this time select **XMLModel Instance**.
4. If you want to, drag and resize the views.

**Tip:** You can save this combination of views as a user-defined perspective. The advantage of doing this is that you can easily alternate between the **Designer** perspective (that has XForms views) and this perspective (that has XML Model views). For detailed information about creating a user-defined perspective, see "Creating user defined perspectives" on page 182.

---

## Creating an XML Model

When creating an XML Model, it is a good idea to create each data instance, along with its associated bindings and submission rules, in turn, following these steps:

1. **Define the data instance** — The first stage is to define the structure of the data instance. In this stage, you define the structure of the data instance by adding elements and attributes to it. You must thoroughly understand XML and the data structure you want to create to complete this stage.
2. **Bind the data instance elements to the form** — The second stage is to bind the data instance to the form. This maps individual data elements to one or more form items, so that they share data.
3. **Set the submission rules** — Finally, you can define the submission rules for the instance if you intend to submit the data separately. These rules determine whether the form is filtered, and sets other submission-related properties.

## Adding an XML Model to a form

In the XML Model view, right-click **No XML Model Exists** and click **Adds XML Model to the form**.

The Designer adds an XML Model and a data instance to the form.

## Adding data instances to the XML Model

When you add an XML Model to a form, by default a data instance is also added. Depending on what data you want the form to collect and submit, you might need to add more data instances to the XML Model. For example, one data instance can contain all the user-inputted information that is to be submitted, while another data instance can contain the user preference data.

To add another data instance to the XML Model:

In the XML Model Instance view, right-click and click **Create XML Model Instance**.

The Designer adds an instance to the XML Model.

Since your form now has more than one data instance, you must name the new instance by giving it a unique **id**. For detailed information about naming data instances, see “Naming data instances.”

## Naming data instances

You name a data instance by giving it an **id** value.

The form’s first data instance is the default data instance and does not require an **id**; however, any additional data instances must have a unique **id**. An instance **id** cannot contain special characters (such as spaces, <, >, &, and so on).

To name a data instance:

1. In the XML Model view, choose the data instance you want to name.
2. In the Properties view, expand **Miscellaneous**.
3. Click within the **id** value field.
4. Type the name and press Enter.

## Deleting a data instance

To delete a data instance:

In the XML Model Instance view, right-click the data instance you want to delete and click **Deletes the selected element**.

---

## Designing data instances

You design XML Model data instances in the XML Model Instance view.

**Note:** When you create an XML Model data instance, the instance is prefixed by `xforms`. For example: `xforms:instance ID =instance1`. It is important to note that this prefix standard was added to XML Model instances while W3C XForms was in development; it is not the official W3C XForms 1.0 standard.

Prior to building your data instance, you should fully map the elements.

For example, if your form has two fields, a popup and a button:

```
- <instance id = yourInstanceName>
  - <document> node
    - form <global> node
      - <page> node
        - <field1> node
          - <value> node
        - <field2> node
          - <value> node
        - <popup> node
          - <popup result> node
```

**Note:** The button does not need to be mapped because it does not collect data from the user.

## Adding elements to a data instance

To build your model, you add elements and attributes to the data instance.

To add an element to a data instance:

In the XML Model Instance view, right-click the data instance and select **Add Element**.

An element — named **instance1** — is added to your data instance.

Next, you would typically rename the element. Once you have done this, you can continue to build the data instance by adding more elements to the data instance, adding elements to this element or adding attributes to this element.

## Renaming data instance elements

When you add an element to a data instance, the Designer generates an auto-incremental element id. To make the elements of the data instance more meaningful, you should rename the element's **id** property value.

**Note:** An element **id** cannot contain special characters (such as spaces, <, >, &, and so on).

To rename an XML Model data instance element:

1. In the XML Model Instance view, right-click the element and click **Rename**.
2. Type the value and press Enter.

## Adding child elements to an element

You can continue to build your model by adding elements to other elements. An element that is added to another element is considered a child element, while the element that has the element added to it is known as the parent element.

To add a child element to a parent element:

1. In the XML Model Instance, right-click the element that you want to add the child element to and click **Add Element**.

A child element is added to the parent element.

2. Repeat this process for every child element you want to add to the parent.

Now, you can continue to build the data instance by adding more child elements to this child element, adding attributes to this child element or naming this child element.

## Deleting data instance elements

To delete a data instance element:

In the XML Model Instance view, right-click the element you want to delete and click **Deletes the selected element**.

## Changing the namespace of data instance elements

If you want to validate an element against a certain schema or your server expects the element to be in a certain schema, you can change the element's namespace.

To change the namespace of an element:

1. In the XML Model Instance view, right-click the element whose namespace you want to change and click **Change Namespace**.  
The schemas you can choose are listed.
2. From the list, select the namespace.

## Adding attributes to data instance elements

To add an attribute to a data instance element:

In the XML Model Instance view, right-click the element that you want to add an attribute to and click **Add Attribute**.

Now, you can either rename the attribute or add a value to the attribute.

## Renaming attributes

To rename an attribute:

1. In the XML Model Instance view, right-click the attribute you want to rename and click **Rename**.
2. Type a name for the attribute and press Enter.

**Note:** An attribute **id** cannot contain special characters (such as spaces, <, >, &, and so on).

## Adding a value to an attribute

To add a value to an attribute:

1. In the XML Model Instance view, double-click the attribute that you want to add a value to.
2. Type a value and then press Enter.

## Converting an attribute to a namespace attribute

If you want to define a namespace to create elements in that namespace, you can convert an attribute into a namespace attribute.

To convert an attribute to a namespace attribute:

In the XML Model Instance view, right-click the attribute you want to convert and click **Convert to Namespace Attribute**.

---

## XML Model binding

Once you have completed modelling the data instance, each data node in the data instance can be mapped to one or more XFDL items in the form. This creates a bind between the data node and the XFDL item, ensuring that they are synchronized; if the value of one changes, the other bound elements are updated to reflect the changes.

## Binding a data instance node to the form

To bind a node to the form:

1. In the XML Model Instance view, select the data node (element or attribute) you want to bind.
2. Drag it to the XFDL item that you want to bind.

The form item is now bound to the data node.

To view the bind:

In the XML Model view, expand **bindings**.

Once you have bound the data nodes to the form items, you can set the submission rules if you want to submit the data instance separately.

## Deleting the bindings for an instance

To delete a binding:

In the XML Model view, right-click the binding you want to delete and click **Deletes the selected element**.

---

## XML Model submissions

When submitting a form that contains an XML 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.

There are many cases in which you may want to submit the entire form, and then retrieve the data instance from the form during processing. This is particularly true when you are using signatures on your forms.

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 data instance may have an associated set of submission rules.

## Adding submissions to an XML Model

To add a submission to an XML Model:

1. In the XML Model view, right-click **XML Model** and select **Create Submissions**.
2. Right-click **submissions** and click **Create submission**.

The Designer adds a submission to your XML Model.

Now, you can name the submission's rules by setting the submission properties.

## Setting the submission rules

The submission rules define what data is submitted, how the data is submitted, and where the data goes. You define submission rules by setting the submission's properties.

To set the submission rules:



1. In the XML Model view, select the submission whose rules you want to set.
2. In the Properties view, expand **Miscellaneous**.
3. Set the submission's properties. For detailed information about submission properties, see "Submission properties."

## Deleting submissions

To delete a submission:

In the XML Model view, right-click the submission you want to delete and click **Deletes the selected element**.

## Submission properties

You name the submission rules by setting the submission's properties. Once set, the rules determine what data is submitted, how the data is submitted, and where the data goes. The submission properties are the following:

**id** Optional: Names the submission rule. If your form has more than one submission, the **id** must be unique. When you add a submission to the XML model, the Designer auto-generates an **id**, however, you should give the rename the **id** a descriptive name to avoid confusion.

**instanceid**

Optional: Defines the instance to be submitted. By default, the first data instance in a form is submitted. If you want to submit a different data instance, you must define that instance using this property.

**action** Optional: Defines the target URL for the submission. You can only list one URL in the action attribute. If you do not provide an action attribute, the submission is sent to the first URL listed in the **url** property of the linked submission button.

**mediatype**

Optional: Sets the content type of the HTTP submission. If you do not provide a media type, it defaults to application/xml.

**includenamespace**

Optional: Restricts the inherited namespaces that are included in the submission. By default, when you submit a data instance, the instance includes all of the namespaces that it inherits. If you want to restrict the data instance's inherited namespaces, add them to this property.

**ref** Optional: Determines which portion of the instance is submitted. You can also choose to submit the entire data instance or only a portion of the instance. When submitting only a portion of the data instance, you must use the **ref** property to identify the root element of the submission.


## Adding an XML submission button

You must create an XML Model before you can set up an XML submission button.

A submission button is only necessary if you want to submit a data instance without the rest of the form. The button triggers the submission, and the data submitted is determined by combining the filters for the button with the submission rules in the data model.

To add and configure an XML submission button:

1. Add a button to your form.

2. In the top-right corner of the Properties view, click .
3. Click **Show Advanced Properties**.
4. In the Properties view, expand **General**.
5. Click within the **value** field.
6. Type the value you want to store in the form when the user selects the choice and press Enter. This value will display as the button label
7. Set **type** to **submit** or **done**.
8. Click within the **url** value field.
9. Type the URL to which the data will be submitted. This must be a complete URL, and may use any of the following schemes: http, https, or file.
10. Expand **Transmit**.
11. Set **transmitformat** to application/xml;id="SubmissionID".
12. Within the SubmissionID quotes, enter the id of the submission.  
By including the name of the appropriate submission rules, you link the button to that set of rules.

---

## Customizing the Designer interface

You can customize many Designer elements including:

- Hot keys
- The Palette
- Perspectives
- Exporting custom objects.

For detailed information on customizing the Eclipse Workbench, see *Customizing the Workbench* in the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

---

### Customizing hot keys

For detailed information about customizing shortcut keys, see *Keys* in the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

---

### Customizing the Palette

You can customize the following Palette features:

- The layout of the button icons
- The button icon size
- Add a User Object Library

#### Selecting a Palette layout

The Palette's layout setting controls how the icons and names of the item buttons are displayed in the Palette's libraries. You can choose one of the following layout options:

- **Columns** — Icons and names are displayed in two columns.
- **List** — Icons and names are displayed in a list.
- **Icons Only** — Only the icons are displayed.
- **Details** — Icons, names and descriptions are displayed in a list.

**Note:** The Palette's current layout is marked by a check.

To change the **Palette** layout:

Right-click the **Palette**, point to **Layout** and then choose the desired layout. To revert to the smaller icon size, clear the check box.

#### Using large button icons

You can increase the default button icon size, making the icons easier to see.

To use large button icons:

Right-click the **Palette** and click **Use large icons**.

## Creating your own custom library

You can create your own custom library containing commonly used items to drag and drop onto your canvas. These saved items or group of items are saved as objects and can then be reused in any forms you are developing.

This is useful if you use an item or group of items often and you want to maintain design standards in another form.

1. Click **Windows** → **Preferences** to open the Preferences window.
2. In the left column, expand **Workplace Forms**.
3. Click **Form Object Library**.
4. Click **New** to add **Directories of object items to add to the palette User Object Library**.
5. Browse to the directory of object items you want to use.  
Multiple directories can be used from your local drive and shared network computers.
6. Click **OK**.
  - If you want to change the order of the object libraries displayed in the palette, click **Up** and **Down**.
  - If you want to remove a directory of object libraries listed in the palette, select the directory and then click **Remove**.
  - If you want to restore directories of your object items listed in the palette, click **Restore Defaults**.
7. Click **OK** to close the Preference window.

**Note:** If you currently have a form open, you must reopen that form before seeing the changes in the palette.

### Tips:

If you want to convert these objects into XForms objects, see “Adding XForms support to an existing form” on page 130.

To expand or collapse Palette libraries, see “Pinning Palette libraries” on page 8

For detailed information about exporting objects, see “Exporting objects” on page 183.

---

## Creating user defined perspectives

The Designer’s default perspective is the **Designer** perspective. If you have modified a perspective by adding, deleting, or moving (docking) views, you can save your changes as a user-defined perspective for future use. For detailed information about perspectives, see *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

To create a user-defined perspective:

1. Modify the Designer perspective by adding, deleting or moving views.
2. Select **Window** → **Save Perspective As**.
3. Name the perspective and then click **OK**.

The Designer perspective was designed to include the views and editors that are required for form design using XFDL and XForms. If you use XML model, you could create an XML Model perspective that includes the XML Model and XML Model Instance views but excludes the XForms and Instance views.

---

## Exporting objects

You can save an item or group of items as an object. You can then reuse that object in any forms you are developing, simply by importing the object into your form. This is useful if you use an item or group of items often.

To export an object:

1. On the canvas, select the items you want to export as an object.
2. Click **Edit** → **Export Objects** to open the Exporting Objects window.
3. Click the **Form Object Library Path** field's **Browse** button.
4. Browse to the directory containing the object items. You can use either:
  - local directories
  - shared network directories

**Note:** Users can change what object items are stored in these directories by opening an Explorer window and adding or removing files. These changes are reflected in the Palette library the next time you open a form.

5. In the **Form Object File Name** field, type the name the object you want to export. This name will be placed on the library button.
6. The rest of the Exporting Objects window's fields are optional.
7. Click **OK** to create the object. A new library is displayed on the **Palette** and your new object is available in that library.

**Note:** If you want to use your own icons, use the following specifications:

- Small Icon — 16 x 16 pixels
- Large Icon — 24 x 24 pixels

If you do not specify a **Palette Drawer Name** the object item will be placed into the **User Object Library**.

For detailed information about

For detailed information about creating your own object library, see "Creating your own custom library" on page 182.



---

## Appendix A: Accessibility

The Designer includes a number of features that make it more accessible for people with disabilities.

---

### Keyboard input and navigation

- **Keyboard focus** — The position of the keyboard focus is highlighted, indicating which view window is active.
- **Toolbar icons** — Each view has its own toolbar. All toolbar controls display tooltips. Icon images used in toolbars and menus are consistent.

---

### Keyboard shortcut keys

Use the keyboard to access functions.

In general, keyboard access conforms to standard Microsoft Windows guidelines. Mnemonics for functions such as menu items are underlined; you can access such functions by holding down the Alt key and pressing the underlined letter key. For example, you can open the **F**ile menu from the keyboard by holding down the Alt key and pressing F.

The Designer allows you to customize your keyboard shortcuts by selecting **Windows** → **Preferences** and expanding **General** → **Editors** → **Keys**. A full listing of functions is listed. For detailed information on customizing keyboard shortcuts, see the *Eclipse Workbench User Guide* (**Help** → **Help Contents**).

---

### Features for accessibility display

A number of features enhance the user interface and improve accessibility for users with low vision. These enhancements include support for the following:

- **High-contrast mode** — The Designer supports the high-contrast mode option that is provided by the operating system. This feature supports a higher contrast between background and foreground colors.
- **Font settings** — The Designer inherits the operating systems settings that you specify for the color, size, and font of text in menus and dialog windows.
- **Color settings** — The Designer inherits the operating system settings for view or window toolbar colors.

---

### Accessible documentation

This document is available in XHTML format within an IBM Eclipse Help System (information center) at: <http://publib.boulder.ibm.com/infocenter/wf/v2r6m1/index.jsp>. For detailed information on accessibility features of the information center, see the information center Welcome page.





---

## Appendix B: Options

What is referred to as a *property* in the Designer, is referred to as an *option* in XFDL.

For detailed information about all options, see the *Workplace Forms XFDL Specification* document.

### **acclabel**

Defines a message that is available to active screen readers. When the focus shifts to the item containing the acclabel, the message is read aloud by the screen reader. The message should contain additional information about the item to assist users with vision impairments. The text entered in this option will not be displayed in the Viewer.

### **activated, focused, and mouseover**

Are not usually declared by the form developer. Instead, they are set for each item by forms viewer software based on system events. Under certain circumstances activated and focused can be set by the form developer.

**active** Specifies whether an item is active or inactive. Inactive items do not respond to user input and, if possible, appear dimmed. For example, an inactive check box will be dimmed and the user will not be able to select or deselect the box.

### **bgcolor, fontcolor, labelbgcolor, and labelfontcolor**

Specifies the colors for an item or its label using either predefined names or RGB (Red Green Blue) triplets.

### **border and labelborderwidth**

Defines whether an item or its label is displayed with a border. Borders are drawn as a three dimensional effect.

### **colorinfo**

Records the colors used to draw the form when a user signs it. This option is only created if the user is allowing the operating system colors to override the color settings in the form. This is most common for users with vision disabilities who may set the operating system colors to provide better contrast between elements on the screen. When the operating system colors override those set by the form itself, it is useful to create a record of those colors so that the appearance of the document, when signed, can be recreated.

### **constraints**

This property group can be used to force user input to meet certain criteria. For example, you can specify that an entry must fall within a certain range, be a certain length, match a template you create, or be in the appropriate case.

### **coordinates**

Records the position of the mouse pointer on an image. The image must exist in a button item. The recording occurs when a user selects (i.e. clicks) the button using the mouse pointer. The position is an intersection on an unseen grid overlaying the image. The points along each axis of the grid range from zero (0) through 1000 with position 0, 0 occurring in the button's top left corner. The coordinates map the intersection closest to the mouse pointer's position.

**data** Stores an information object such as an image, a sound, or an enclosed file in an XFDL form. Whenever any of these objects are added to a form, the data that describes the object is stored in a data item. A data item can only store the data from a single object. Data in data items must be encoded in base64 format. Data items are created automatically when files are enclosed in a form. Enclose files using items with a type option setting of `enclose`.

**datagroup**

Provides a way of associating related data items to each other and to certain other items. There are two ways of using this option. In the first case, it enables you to create a group of data items, called a datagroup. In the second case, this option enables you to reference such a datagroup from button, action, or cell items. This option is most often used to group file enclosures. For example, you can use this feature to create folders with which users can organize their enclosures. Each enclosed file can belong to several datagroups, and each datagroup can contain several enclosed files.

**datatype**

This property determines what kind of information the item should accept. For example, if you set a field's **datatype** to `Dollar`, then it will accept only dollar values (whole or decimal numbers) as input.

**delay** Delays the execution of an automatic action or specifies an automatic action repeat factor. Repeated actions stop when the page containing the action definition closes. Define automatic actions using an action item.

**dirtyflag**

Alerts the form viewing program to the changed status of the form. Dirtyflag 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.

The dirtyflag is set to on whenever the user makes a change to the form. Such changes include typing information into the form, selecting choices in lists or with radio buttons, and so on. The dirtyflag is set to off whenever the user saves or submits the form.

Note that the dirtyflag is not set by computed changes to the form. For example, if the user clicks a button that triggers a compute, and that compute copies information to a field in the form, the dirtyflag would not be set. In these cases, the form should include additional computes that set the dirtyflag.

If necessary, the save prompt can be disabled by using a compute to set the dirtyflag to off.

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**excludedmetadata**

This option allows additional data about a signature to be included, but never signed. This makes it possible to store the notarization of signatures without interfering with other, overlapping signatures.

For example, if Signer1 signs a form and then Signer2 affixes an overlapping signature, you could not modify the first signature without breaking the second. In this case, you would not be able to notarize the first signature, since affixing the notarization would change the metadata of that signature and break the second signature.

The `excludedmetadata` provides a place to store the notarization for the first signature without breaking the second signature. You can add information to this option at any time, since the `excludedmetadata` option is never signed.

**filename**

Identifies the name of an enclosed file. This name appears in the list of enclosed files.

**focused**

Specifies whether an item, page, or form currently has the input focus. This option is usually set by code outside XFDL, but can also be set by a `compute`, provided that the `compute` is setting the focus of an item to on, the item is on the same page, and the item receiving the focus is capable of doing so. This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**focuseditem**

Specifies which item in the page currently has the focus. This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**fontinfo and labelfontinfo**

Defines the character set, font name, point size, and font characteristics for the text portion of an item's value or label. Note that the font selected for an item influences the item's size.

**Note:** If you set the `fontinfo` option using the Font properties window, setting Color or Strikethrough in the window will have no effect. To set the font color, use the `fontcolor` option. Strikethrough text is not supported by XFDL.

**format**

Allows edit checks and formatting options for field, label, list, popup, and combobox items to be specified. It allows a mandatory status for signature button items to be specified. For detailed information about button item descriptions, see the *Workplace Forms XFDL Specification* document.

**formid**

Defines a unique identifier for the form, such as a serial number.

**fullname**

Used in a signature item to record the fully qualified name of the signer. This name is retrieved from the digital certificate used to sign the form.

**group**

Provides a way of associating related items. There are two ways of using this option. In the first case, it enables you to create groups of cells or radio buttons. In the second case, the `group` option enables you to populate lists, popups, and comboboxes by referencing a group of cells. Items with the same `group` reference are considered members of the same group.

**help**

Lets you create context-sensitive help for each visible item in a form. When a user opens the form in the Viewer and moves their mouse pointer over the item, the help message will appear. The item reference identifies the help item containing the help message. There can be many items pointing to the same help message.

**image**

Associates an image with an item. The item reference identifies the data item containing the image. This image replaces any text label if the viewer is able to display images.

**imagemode**

Defines how the image will be displayed in the item. The image may be clipped, resized, or scaled to fit the item.

**itemfirst**

Identifies the first item on the page, excluding the global item. An item is first when it appears first in the build order (in other words, it is first in the XFDL text). This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**itemlast**

Identifies the last item on the page, excluding the global item. An item is last when it appears last in the build order (in other words, it is last in the XFDL text). This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**itemlocation**

Serves two purposes:

- It specifies the location of an item in the page layout.
- It lets you set the size of the item, either in relation to another item, or in absolute terms.

Itemlocation offers three ways to position items on the page: absolute positioning, relative positioning, and offset positioning. Absolute positioning anchors the top left corner of an item to a particular location on the page, using an x-y coordinate. For example, you might place an item 10 pixels in from the left margin, and 10 pixels down from the top of the page. Relative positioning places items on the page in relation to one another. For example, it might place one item below another. Finally, offset positioning allows you to place an item on the page relative to another item, and then move it a set amount. For example, you might place an item below another, and then move it 10 pixels to the right.

Itemlocation also provides two ways to set the size for an item: relative positioning and extent sizing. Relative positioning allows you set the size of an item relative to another item on the page. For example, you might expand an item so that its right edge lines up with the right edge of a different item. Extent sizing allows you to set the absolute size of an item in the pixels. For example, you might set an item to be 100 pixels wide and 30 pixels tall.

Note that you can also combine these methods for positioning and sizing. For example, you might place an item on the form using absolute positioning, and then place a second item below the first using relative positioning.

**itemnext**

Identifies the next item on the page, excluding the global item. An item is next when it appears next in the build order (in other words, it is next in the XFDL text). This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**itemprevious**

Identifies the previous item on the page, excluding the global item. An item is previous when it immediately precedes the current item in the build order (in other words, it comes immediately before the current item in the XFDL text). This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**Note:** Avoid using a compute that references *itemprevious* from within an item that directly follows a container item (for example, a pane or table) or a controlled item (for example, a checkgroup or radiogroup) in the form's build order. Instead, use an explicit reference to the item. If you do use an *itemprevious* compute in this situation, the Designer will not evaluate the compute in the same way that the Viewer will.

**justify**

Controls whether text in the item should be left, center, or right justified.

**keypress**

Contains the last keystroke made by the user in the focused item, page, or form. A keypress option is ignored if no keypress has been established at the level of focus. If the value of a keypress option is ignored at the item level, it passes up to the page level, and if ignored at the page level, it passes up to the form level. This option allows for the creation of a default button (shortcut key) on a page or a form. This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**label** Defines a static text message or an image to display on the form. If both an image and a text message are defined for the label, the image takes precedence in viewers able to display images.

**labelborder**

Defines whether there is a border around the label specified in the label option.

**last** Identifies the last item in a repeat, group, or switch. This is the item that receives the focus when the user tabs backward into a group, a particular case, or a new row in a repeat.

This option affects the tab order in the following ways:

— When the user tabs backward into a table or pane, the focus goes to this item. In the case of a table, the focus goes to this item in the last row.

— When the user tabs backward from the beginning of a row, the focus goes to this item in the previous row or to the item that precedes the table or pane.

— When the user tabs forward from this item, the focus goes to the next row or to the item that follows the table or pane.

**layoutinfo**

This option records location information for all visible signed items. A hash is taken of each page containing a signed item, and this hash includes positioning information for all the signed items relative to each other in those pages.

**linespacing**

This option adjusts the spacing between lines of text. This sets an offset value, which will add to or subtract from the default spacing. For example, a value of 1 will add one pixel to the space between each line, while a value of -1 will remove one pixel from the space between each line.

**mimedata**

Contains the actual data associated with a data item or a signature item. It can be binary data or the contents of an enclosed file. The data is encoded in base64 format, so that even forms containing binary data can be viewed in a text editor. When the data is needed by the form, it is decoded automatically from base64 back to its native format. Data may also be compressed before base64 encoding, allowing an item to store a larger block of data.

**mimetype**

Defines the MIME type of the data stored in a data item.

**next**

Identifies the item to receive focus when a user tabs ahead from the current item. If a user tabs ahead from the last item on the page, the tab cycles within the same page, beginning with the first item on the page. Only modifiable or read only items can receive focus.

**pageid**

Defines a unique identifier for a page, such as a serial number.

**pagefirst**

Stores a reference to the global item on the first page of the form, excluding the global page. A page is first when it appears first in the build order (in other words, it is first in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**pagelast**

Stores a reference to the global item in the last page of the form, excluding the global page. A page is last when it appears last in the build order (in other words, it is last in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**pagenext**

Stores a reference to the global item in the next page in the form, excluding the global page. A page is next when it appears next in the build order (in other words, it is next in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**pageprevious**

Stores a reference to the global item in the previous page in the form, excluding the global page. A page is previous when it immediately precedes the current page in the build order (in other words, it is immediately previous in the XFDL text).

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**presentation**

This property group formats how the data is displayed. For example, if the data type is Dollar, you can set the presentation property to add a dollar sign.

**previous**

Identifies the item to receive focus when a user tabs backwards, using Shift+ Tab, from the current item. If the current item has a previous option, the item indicated in that option is next in the reverse tab order. If the current item has no previous option, the previous item in the build order that can receive the input focus is next in the reverse tab order.

**printbgcolor**

Enables the form to be printed with a specific background color on a color printer. This color can be the same as or different from the background color shown on the screen. On black and white printers, grayscaling is used.

**printing**

Indicates whether the form is currently printing. This value toggles from off to on just before printing. Any computes that rely on this option are updated before the form prints. This allows you to make computed changes to the form just before it is printed.

This option is not saved or transmitted as part of the form. Instead, it is automatically created each time the form is read into memory, and is maintained only during display or processing.

**printfontcolor**

Enables the item to be printed with a specific font color on a color printer. This color can be the same as or different from the font color shown on the screen. On black and white printers, grayscaling is used.

**printlabelbgcolor**

Enables an item's built-in label to be printed with a specific background color on a color printer. This color can be the same as or different from the background color shown on the screen. On black and white printers, grayscaling is used.

**printlabelfontcolor**

Enables an item's built-in label to be printed with a specific font color on a color printer. This color can be the same as or different from the font color shown on the screen. On black and white printers, grayscaling is used.

**printsettings**

Determines the settings that will be used when the form is printed. The user can be allowed to change these defaults, or the form can be set so that it will always follow the defaults.

**printvisible**

Determines whether an item should be visible when the form is printed. Has no effect on the visibility of the item on the screen.

**readonly**

Sets the item to be read only, so that users can read information in the item but cannot change that information.


**requirements**

Specifies one or more requirements that must be satisfied before the form will function properly. For example, a form may require a Java Virtual Machine to run correctly. You can use the requirements feature to check for



the availability of a particular class, function call, or Java Virtual Machine. If the requirement is not met, the user will receive a customizable error message that explains the problem.

**rtf** Stores the rich text value for rich text fields.

**Note:** Do not use the editor associated with the *rtf* property/option (that is, within the Properties view, the editor displayed when you click  within the *rtf* field). The *rtf* option is for storing the rich text information entered by the end-user. Do not set this property/option in the Designer.

#### **saveformat**

Specifies the format a form will be saved in. An XFDL form may be saved in XFDL format or HTML format. Furthermore, the XFDL format may be compressed using ASCII compression. The formats work as follows:

— XFDL format saves the entire form definition, including the user input.

— HTML format saves the form as a series of assignment statements for each modifiable item, equating the item reference with the item's value. The only items included in the save are custom items and the following modifiable items: check, field, list, popup, combobox and radio.

#### **scrollhoriz and scrollvert**

Control whether a text field item has horizontal and vertical scrollbars or whether it wordwraps, allows vertical sliding, and so on.

#### **signature**

Contains a signature and the data necessary to verify the authenticity of a signed form. It is created by a form viewer or other program when a user signs a form (usually using a signature button). The signature item contains an encrypted hash value that makes it impossible to modify the form without changing the hash value that the modified form would generate. To verify, one can generate the hash value and then see if it matches the one in the signature.


#### **signatureimage**

Points to a data item, identifying it as the data item into which the captured signature image is placed. Used only with image-based digital signatures (such as Silanis).

#### **signdatagroups**

Specifies one or more datagroups, as defined by the *datagroup* option, that the signature will either keep or omit. 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.

**signdatagroups** has the following properties:


- **filter** — Specifying **keep** will include datagroups in the **datagroup** reference list (**Refs**) with the signature and omit those that are not in the list. Omit will not include the datagroups specified in the **datagroup** references list from the signature; however, it will include all of those that are not in the **Refs** list.
- **Refs** — A string that identifies a **datagroup** whose data items will be filtered. In the Properties view, click  to add a **datagroupref** item.



## signdetails

Specifies which certificate attributes are shown to the user when they are choosing a certificate to sign the form, and defines the filters used to select the available certificates when the user is signing a form. For example, the signdetails option could specify that only those certificates with a common name that begins with "Bob" are shown, and that only the owner's common name and e-mail address are shown.

**signdetails** has the following properties:

- **datacolumns** — Specifies a list of certificate attributes that the user sees as they select a certificate to sign a form.
- **filteridentity** — Specifies a list of certificate attributes (**tag**) and values (**value**) that filters certificates to determine which certificates are available to the user for signing purposes. In the Properties view, click  in the **Value** column for **Filter List** property to add a **filter** property.

**tag** is a string that contains the names of the attribute you want to use to filter the available certificates. **Value** is a string used to compare with the tag. Use an asterisk character ("\*") as a wildcard for multiple characters and a question mark character ("?") as a wildcard for a single character.

The following is a list of case-sensitive attributes for common certificates:

- **Version** — The version of the specification that the certificate follows.
- **Serial** — The certificate's serial number.
- **SignatureAlg** — The algorithm used by the certificate Authority to sign the certificate.
- **BeginDate** — The date the certificate became valid.
- **EndDate** — The expiry date for the certificate.
- **PublicKey** — The certificate's public key.
- **FriendlyName** — The certificate's friendly name.
- **Subject: CN** — The certificate owner's common name.
- **Subject: E** — The certificate owner's e-mail address.
- **Subject: T** — The certificate owner's title.
- **Subject: L** — The certificate owner's locality.
- **Subject : ST** — The certificate owner's state of residence.
- **Subject: O** — The organization to which the certificate owner belongs.
- **Subject: OU** — The name of the organizational unit to which the certificate owner belongs.
- **Subject: C** — The certificate owner's country of residence.
- **Subject: STREET** — The certificate owner's street address.
- **Subject: ALL** — The certificate owner's complete distinguished name.
- **Issuer: CN** — The certificate issuer's common name.
- **Issuer: E** — The certificate issuer's e-mail address.
- **Issuer: T** — The certificate issuer's title.
- **Issuer: L** — The certificate issuer's locality.
- **Issuer: ST** — The certificate issuer's state of residence.
- **Issuer: O** — The organization to which the certificate issuer belongs.
- **Issuer: OU** — The name of the organizational unit to which the certificate issuer belongs.
- **Issuer: C** — The certificate issuer's country of residence.
- **Issuer: STREET** — The certificate issuer's street address.

- Issuer: ALL — The certificate issuer’s complete distinguished name.

**signer** It is automatically generated and records the identity of the person who signed the form. The setting of the signer option varies according to the engine type used.

- ClickWrap — The signer setting uses Accepted.
- CryptoAPI — The signer setting uses common name and e-mail.
- Entrust — The signer’s login identity.
- Generic RSA — The signer setting uses common name and e-mail.
- HMAC-ClickWrap — The value of the answer indicated by the HMACsigner tag in the signformat option.
- Netscape — The signer setting uses common name and e-mail.
- signaturePad
- Silanis

**signformat**

Sets the details of the signature, including the **mimetype** to encode it, the signature engine to create it, and special settings for the signature engine.

*MIMEtype;engine;verifier;cval;delete;parameters*

**MIMEtype**

Required. The MIME type used to store the signature information. Typically, you should use **application/vnd.xfdl**.

**engine**

Required. The name of the signing engine to use (the default is Generic RSA if nothing is specified). The types of signature engines you can specify are: ClickWrap, CryptoAPI, Entrust, Generic RSA, HMAC-ClickWrap, Netscape, signaturePad, and Silanis. The Generic RSA signature engine includes CryptoAPI and Netscape. The HMAC-ClickWrap refers to the Authenticated Clickwrap signature engine.

**verifier**

Optional. A string that indicates which identifier to use when verifying certificate chains during digital signature operations: Basic or DODJ12.

- Basic — Performs basic certificate verification. Basic is the default.
- DODJ12 — Performs strict certificate verification that complies with the US Department of Defense requirements.

**cval**

Optional. Indicates whether the current value of computed operations on the form are signed. Use this parameter when you want to sign formulas, but not the value calculated by the formula. The default is that they are signed (“on”). If you do not want them signed, set this parameter to “off”.

**delete**

Optional. This flag sets whether the user can delete the signature. By default, users can delete all signatures (“on”). If you want to prevent a signature from being deleted, set this to “off”.

**parameters**


Depending on the signature engine you specify, your engine may include additional parameters.

**signgroups**

Specifies one or more groups, as defined by the group option, that the

signature will either keep or omit. This filters any radio buttons or cells belonging to that group, but does not filter list, popup, or combo box 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.

**signgroups** has the following properties:


- filter — Specifying **keep** will include groups of cells in the **groupref** reference list (**Refs**) with the signature and omit those that are not in the list. Omit will not include the grouprefs specified in the group references list from the signature; however, it will include all of those that are not in the **Refs** list.
- Refs — A string that identifies a group whose cell items will be filtered. In the Properties view, click  to add a **groupref** item.

### **signinstance**

Specifies what XForms instance data is filtered for a particular signature. Filtering instances means keeping or omitting specific data from each data instance. For example, you might set the filter to omit any data that is not sent to the server.

When instance data is omitted from a signature but the associated user interface elements are signed, the user can still enter data into those elements. Furthermore, the overlap and layout tests are not performed on those items. This leaves them free to change certain characteristics, such as size (for expanding tables or fields), to accommodate the user input. This facilitates signing the presentation layer of a form while leaving the actual data open to change.

**signinstance** has the following properties:

- filter — Specifying **keep** will include groups of cells in the **Dateref** reference list (**Refs**) with the signature and omit those that are not in the list. Omit will not include the groups of cells specified in the references list from the signature; however, it will include all of those that are not in the **Refs** list.
- Refs — A string that identifies a **type** (element tag name) of items that will be filtered. In the Properties view, click  to add an **itemtype** item. This data reference has a model ID and an XPath. The XPath reference is to the root of the data where to want to filter.

### **signitemrefs**


Specifies individual items that are filtered for a particular signature. Filtering an item reference means keeping or omitting specific items, rather than all items of a particular type (see **signitems**). For example, you might set the filter to omit **BUTTON1** on **PAGE1**.

### **signitems**

Specifies which types of items filtered for a particular signature. Filtering an item means keeping or omitting all items of a particular type, rather than specific items (see **signitemrefs**). For example, you might set the filter to omit all button items from the signature.

**signitems** has the following properties:

- filter — Specifying **keep** will include types of options in the **optiontype** reference list (**Refs**) with the signature and omit those options that are not in the list. Omit will not include the optionstypes specified in the optionstypes references list from the signature; however, it will include all of those options that are not in the **Refs** list.

- Refs — A string that identifies a **type** (element tag name) of options that will be filtered. In the Properties view, click  to add an **optiontype** item.

### signnamespaces

Specifies which namespaces are filtered for a particular signature. Filtering a namespace means keeping or omitting all of the form elements and attributes that are in the specified namespace.

For example, if a signnamespaces option specifies that the `http://www.PureEdge.com/XFDL/Custom` namespace should be kept, then all elements in that namespace are signed.


**signnamespaces** has the following properties:

- filter — Specifying **keep** includes all form items in the namespace URL list with the signature. It omits all of those items not in the list. **omit** will omit all form items that are not in the namespaces in the namespace URL list from the signature, and it will only include those that are not in the list.
- Refs — A string that identifies a namespace whose items will be filtered.

### signoptions

Specifies which types of options are filtered for a particular signature. Filtering options means keeping or omitting all options of a particular type, rather than specific options (see `signoptionrefs`).


**signoptions** has the following properties:

- filter — Specifying **keep** will include types of items in the **itemtype** reference list (**Refs**) with the signature and omit those that are not in the list. Omit will not include the itemtypes specified in the itemtypes references list from the signature; however, it will include all of those that are not in the **Refs** list.
- Refs — A string that identifies a **type** (element tag name) of items that will be filtered. In the Properties view, click  to add a **itemtype** item.

### signoptionrefs

Specifies individual options that are filtered for a particular signature. Filtering option references means keeping or omitting specific options, rather than all options of a particular type (see `signoptions`). For example, you might set the filter to omit `BUTTON1.value` on `PAGE1`.

**signoptionrefs** has the following properties:

- filter — Specifying **keep** will include options in the **optiontype** reference list (**Refs**) with the signature and omit those that are not in the list. Omit will not include the optiontypes specified in the option types references list from the signature; however, it will include all of those that are not in the **Refs** list.
- Refs — A string that identifies the option to be filtered. In the Properties view, click  to add a **Refs** item.

### signpagerefs

Specifies individual pages that are filtered for a particular signature. Filtering pages means keeping or omitting a page and all of its contents.

**signnamespaces** has the following properties:

- filter — Specifying **keep** includes all pages in the page reference list with the signature. It omits all of those items not in the list. **omit** will omit all pages that are not in the page reference list from the signature, and it will only include those that are not in the list.

- Refs — A string that specifies the page to be filtered.

**size** Specifies an item's size. It does not include external labels, borders, or scroll bars. These are part of the bounding box size which is calculated automatically. Examples of item size are the input area in a field item or the height and width of the label in label and button items.

**suppresslabel**

Suppresses the built-in label for some items, so that the label is not shown even if the label option or xforms:label option is set.

This is most useful when you are using XFDL to wrap an XForm control that includes labels that are not necessary in the visual presentation. For example, you might not want to display the labels of items in a table.

When the label is suppressed, the item is both displayed and printed as if no label were present at all. This means that both the appearance and size match an equivalent item with no label.

**texttype**

Sets whether a field contains plain text or rich text

**thickness**

Specifies the thickness of a line item. The unit of measurement is pixels.

**transmitformat**

Specifies the format of the form data submitted to a processing application. An XFDL form can submit data in XFDL format or in HTML format. Furthermore, the XFDL format may be compressed using ASCII compression.

XFDL format submits the entire form definition, including user input.

HTML format submits just an assignment statement for each item equating the item reference with the item's value. The only items included are modifiable items, custom items, and items with a transmit option setting of all.

**transmitdatagroups, transmitformat, transmitgroups, transmititemrefs, transmititems, transmitnamespaces, transmitoptionrefs, transmitoptions, and transmitpagerefs**

Work together to allow you to transmit form submissions.

**triggeritem**

Set in the form globals to identify which action, button, or cell activated a form transmission or cancellation.

**type** Specifies whether the action, button, or cell item will perform a network operation, print, save, digitally sign, and so on.

**url** Provides the url to a target, such as a file or application. Items containing this option must have a type option setting of link, replace, submit, done, or pagedone.

The object identified must be one of the following:

File — Used with a type option of link or replace. The file identified is downloaded, and either displayed or saved. Examples of such files are images, word processing documents, and XFDL forms.

Application — Used with a type option of submit or done. The application identified is initiated. A form processing application, such as a cgi or a servlet, is an example of such an application.

Item — Used with a type option of pagedone. The item identified, on the page identified, receives focus. The item must be on another page.

Form or Page Globals — Used with a type option setting of pagedone. The focus moves to the first item on the page when the new form or page appears. The form globals reference is global.global. The page globals reference is <page sid> global for another page

E-mail Address — Used with a type option of submit, done, link, or replace. With a submit or done type, the form is attached to an e-mail message, and that message is sent to the address in the url. With a link or replace type, an e-mail message is created and sent, but the form is not attached to the message. Depending on the settings you use, the user may be able to add additional information to the e-mail.

**value** Reflects the contents of an item. Visually, this can take several forms, depending on the item to which it applies. For example, the value option in label items contains the label text; the value option in radio items contains a status indicator; and the value option in list items contains the scope identifier (sid) of the most recently selected cell (if it was a select cell). An item's contents will be stored in the form whenever a user saves the form or submits it for processing. This is true even for inactive items and items using the default value option setting (in this case, a value option containing the default setting is added to the item's definition).

**visible**

Determines whether the item should be shown to the user or made invisible.

**webservices**

Defines the name of the Web services used by the form.

**writeonly**

Sets a field to be write only. This means that the user can type into the field, but cannot read what is typed. Instead, each character is replaced by a uniform symbol (such as an asterisk).

This is useful if you are creating a password field.

**custom option**

Allows form designers to add application specific information to the form definition. This is useful when submitting forms to applications requiring non-XFDL information. An example of non-XFDL information might be an SQL query statement. Custom options must not be in the XFDL namespace.

---

## Appendix. 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.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

**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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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.

---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX  
IBM  
Workplace  
Workplace Forms

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.



---

# Index

## A

- accessibility
  - adding help messages 55
- actions 67
  - automatic 71
  - delete 158
  - insert 156
  - setindex 156, 157
  - submissions (XML Model) 179
  - toggle 151
- Add Element (XML Model) 176
- adding
  - a help message (XForms) 160
  - a hint message (XForms) 160
  - a value to an attribute (XForms) 137
  - an alert message (XForms) 159
  - attributes to an element (XML Model) 177
  - attributes to elements (XForms) 137
  - bind to a model (XForms) 164
  - cases to a Switch 149
  - child elements (XForms) 136
  - child elements to an instance (XML Model) 176
  - Choice (Item) 145
  - data instances (XML Model) 174
  - elements to an instance (XML Model) 176
  - items (XForms), best practices 138
  - items to a case (XForms) 150
  - items to a table (XForms) 155
  - list items to List (Select) 146
  - list items to List (Select1) 146
  - sibling elements (XForms) 136
  - submissions (XForms) 165
  - submissions (XML Model) 178
  - Switch 149
  - table row (XForms) 156
  - text value to elements (XForms) 136
  - value to an attribute (XML Model) 177
  - Web services to form 127
  - XForms 130
  - XML Model 174
- advanced properties 53
- alert message (XForms)
  - adding 159
- aligning
  - aligning items 43
  - bounding box 37
  - expanding items 46
  - removing relative positioning 46
  - snapping to items 39
  - snapping to the grid 39
  - types defined 44
- Appearance properties
  - visibility on/off 48
- approval signatures 120
- attachments 105
  - buttons 106

- attributes
  - nodes (XForms) 134
- attributes (XForms)
  - adding a value 137
  - overview 134
- attributes (XML Model)
  - adding to a value 177
  - adding to an element 177
  - binding 178
  - converting namespace 177
  - renaming 177
- audio help
  - adding messages 55
  - Help properties 54
- Authenticated Clickwrap signatures 112
- Authenticated Clickwrap signatures button 115
- automatic actions 71

## B

- background images 83
- best practices
  - adding items (XForms) 138
  - binding items (XForms) 138
- bind
  - adding to a model 164
  - binding (XForms) 162
- bind property (XForms binding)
  - overview 162
- binding
  - attribute (XML Model) 178
  - elements (XML Model) 178
  - items (XForms), best practices 138
  - label property (XForms) 140
  - nodes (XML Model) 178
  - repeat item 154
  - using the bind property (XForms) 162
  - using the nodeset property (XForms) 162
  - using the ref property (XForms) 161
  - XForms label property 140
  - XForms lists to a node set 143
  - XForms overview 161
  - XForms table 154
  - XML Model 177
- binding (XML Model)
  - deleting 178
- Bitmap 81
- bound items (XForms)
  - highlighting 164
- bounding box
  - aligning items 37
- build order
  - changing order 42
  - pasting items 48
- building
  - data instances (XForms) 134
  - data instances (XML Model) 175
- buttons 67

- buttons (*continued*)
  - about radio buttons 77
  - adding images 82
  - attachment 106
  - compress XFDL file 68
  - create radio button (XFDL) 77
  - delete table row 158
  - image-mapping 83
  - insert table row 156
  - link or replace 71
  - print 71
  - save or cancel 71
  - submission (XForms) 169
  - submission (XML Model) 179
  - submit 68
  - toggle between cases 150
  - types 67

## C

- C language
  - custom functions 101
- Calculate property (XForms) 163
- calendar
  - create 80
- case
  - adding items (XForms) 150
  - adding to a Switch 149
  - Make Active 150
  - setting the default 150
- cell
  - add to radio button (XFDL) 77
- changing
  - namespace (XML Model) 177
  - Palette appearance 181
- check box (XFDL)
  - create 76
  - when to use 75
- check group list
  - displaying list choices 144
- child elements (XForms)
  - adding 136
  - overview 134
- child elements (XML Model)
  - adding to an instance 176
- child nodes (XForms) 134
- choice
  - adding to a CheckGoup 145
  - adding to a RadioGoup 145
  - calendar 80
  - Combobox 78
  - create a check button 76
  - creating box list 79
  - Palette items 75
  - popup lists 77
  - radio button (XFDL) 77
- Clickwrap signatures 111, 114
- Clickwrap, Authenticated 112
- Clickwrap, Authenticated button 115
- clipping images 82

- closing
  - form 21
  - projects 13
  - views 8
- code assist
  - editing source code 7
- color
  - apply to item 53
  - background 52
  - grid 38
  - RGB 52
  - text 52
  - toolbar 52
  - Viewer error color 52
  - Viewer mandatory item color 52
- ComboBox
  - create list 78
- compress
  - XFDL files 68
- compressing
  - forms 19
- Compute Wizard
  - assignment formula 86
  - calculation formula 86
  - calling custom functions 102
  - custom formula 97
  - function formula 88
  - functions defined 91
  - if/then/else formula 89
  - logical operators 90
  - using custom functions 101
- conditional items (XForms)
  - overview 148
- configuring
  - delete table row button 158
  - insert table row button 156
- Constraint property (XForms) 163
- constraints (XFDL)
  - defined 61
  - on data input fields 61
  - specifying data restrictions 61
- context-sensitive help
  - adding to an item 54
  - Help properties 54
- converting
  - namespace attributes (XML Model) 177
  - XFDL items to XForms items 160
- copying
  - data to another field 86
  - items 48
  - property settings 51
- Create Submission (XForms) 165
- creating
  - a table template row (XForms) 155
  - conditional items (XForms), overview 148
  - data instance from schema (XForms) 133
  - data instances (XForms) 132
  - delete table row button (XForms) 158
  - empty data instance (XForms) 133
  - forms 17
  - insert table row button (XForms) 156
  - labels (XForms) 139, 140
  - List (Select1) 146
  - model binds 164

- creating (*continued*)
  - model binds (XForms) 164
  - multi-line fields (XForms) 141
  - password fields (XForms) 141
  - projects 13
  - single line fields (XForms) 140
  - submission button (XForms) 169
  - submissions (XML Model) 178
  - tables (XForms) 151
  - user defined perspective 182
  - XML Model Instance 174
  - XML Model, overview 174
- creating data instances
  - from WSDL message 134
- creating data instances (XForms)
  - from document 132
- cropping images 82
- CryptoAPI, Microsoft 110, 113
- currency
  - formatting types 60
- customizing
  - Designer 181
  - hot keys 181
  - Palette 181
  - perspectives 181
  - shortcut keys 181
- cutting
  - items 48

## D

- data
  - constraints 61
  - copy data to another field 86
  - displaying, submitted 170
  - planning formulas 85
  - reformatting 63
- data instances (XForms)
  - adding submissions 165
  - naming 134
  - overview 131
- data instances (XML Model)
  - adding 174
  - building 175
  - deleting 175
  - naming 175
- data submission (XForms)
  - overview 165
  - using bind property 167
  - using ref property 167
- datagroupref 105
- datatype
  - formatting input fields 59
- date
  - create calendar 80
  - display current date 100
  - formatting types 60
- default case
  - setting 150
- defining
  - absolute align 43
  - absolute expand 46
  - alignment types 44
  - constraints 61
  - datatypes 60
  - expansion types 46
  - formula functions 91

- defining (*continued*)
  - formula operators 96
  - formula reference types 97
  - if/then/else operators 90
  - list choices in the XForms list
    - item 144
  - presentation properties 63
  - relative align 43
  - relative expand 46
  - submissions rules (XML Model) 178
- delete
  - action 158
- deleting
  - binding (XML Model) 178
  - creating a delete table row
    - button 158
  - data instances (XML Model) 175
  - elements from an instance (XML Model) 176
  - forms 22
  - formula 98
  - guides 40
  - items on form 48
  - list items in a Select (List) 147
  - list items in a Select1 (List) 147
  - projects 15
  - selected element (XML Model) 178
  - submission rules (XML Model) 179
  - table row 158
  - WSDL file 127
- descriptions
  - form templates 20
- Designer
  - canvas defined 6
  - customize 181
  - interface, overview 5
  - introduction 3
  - perspectives 5
- designing
  - data instances (XForms), overview 134
  - data instances (XML Model) 175
  - lists tips (XForms) 141
- digital signatures 109
- displaying
  - bound items (XForms) 164
  - list choices from a node 142
  - list choices in check group list 144
  - list items (XForms) 143
  - submission data 170
  - XML Model Instance view 174
  - XML Model views 174
- dynamically displaying
  - list choices 142

## E

- e-mail
  - transmission of forms 68
- editing source code
  - code assist 7
- editors
  - Design 6
  - overview 6
  - Preview 6
  - Source 6

- elements (XForms)
  - adding attributes to 137
  - adding text value to 136
  - removing 136
  - renaming 136
- elements (XML Model)
  - adding to an instance 176
  - binding 178
  - deleting from an instance 176
  - renaming 176
- enclosures
  - creating attachments 105
  - image file 81
  - view overview 10
- encryption 108
- Entrust signatures 110, 116
- error messages
  - creating 55
  - Help properties 54
  - mandatory field 62
  - Problems view 9
- example
  - display current date 100
  - formula 98
  - XForms table 153
- expansion types
  - defined 46
- exporting
  - forms 22
  - objects 183

## F

- field (XFDL)
  - formatting input fields 59
- fields (XForms)
  - overview 140
- file/folder
  - creating a project 13
  - linking to project 14
- filtering
  - submissions (keep/omit) 70
  - types for signature 121
- form
  - adding XForms support 130
  - changing build order 42
  - closing 21
  - compressing 19
  - compression 68
  - creating 17
  - creating project folders 13
  - cutting items 48
  - deleting 22
  - deleting items 48
  - embedding JAR file 102
  - exporting 22
  - importing 22
  - installing IFX files 102
  - opening 17
  - pasting items 48
  - planning formulas 85
  - recover previous version 21
  - renaming 19
  - route 68
  - routing 68
  - saving new name 19
  - setting global properties 19

- form (*continued*)
  - transmission 68
  - upgrading 18
- formatting
  - input data types defined 60
  - mandatory fields 62
  - presentation properties defined 63
  - specifying constraints 61
  - specifying type of data accepted 59
  - visibility 48
- formula
  - advanced calculation 87
  - calculate items 87
  - calculate number or text 86
  - create custom function 97
  - delete 98
  - functions defined 91
  - if/then/else 89
  - if/then/else example 91
  - if/then/else operators 90
  - operators defined 96
  - planning 85
  - reference item types 97
  - sample 98
  - simple 86
  - sum 88
  - value equal to a function 88
  - when to use 85
- functions
  - custom 101
  - custom overview 97
  - custom, planning 101
  - equal to 88
  - IFX install location 102
  - libraries 101
  - using custom functions 102

## G

- Generic RSA signatures 109, 112
- get
  - submission type (XForms) 168
- getting started 3
- GIF 81
- global form properties 19
- grid
  - changing color 38
  - changing size 38
  - layout tool 38
  - showing 38
  - snapping 39
- grouping
  - radio buttons (XFDL) 77
- guides
  - creating 40
  - deleting 40
  - inserting onto a ruler 40
  - moving 40
  - overview 39
- gzip
  - compress XFDL file 68

## H

- help
  - adding 160

- help (*continued*)
  - adding audio help 55
  - adding context-sensitive help 54
  - adding message (XForms) 160
  - creating error messages 55
  - hint message (XForms) 160
  - message types 54
- hiding
  - grid 38
  - items 40
  - page size 40
  - ruler 39
  - the Palette 8
- highlighting
  - bound items (XForms) 164
- hot keys
  - customizing 181
- HTTP
  - transmission of a form 68

## I

- icons
  - closing project 13
  - large Palette 181
  - opening project 13
- id
  - data instances (XML Model) 175
  - element (XML Model) 176
  - renaming attributes (XML Model) 177
  - submissions (XML Model) 179
  - XForms Model 131
- IFX
  - custom functions 101
  - install location 102
- images
  - adding to button 82
  - adding to page 81
  - background image 83
  - changing modes 82
  - clipping 82
  - considerations 81
  - enclosing a file 81
  - enclosing an image file 81
  - form templates 20
  - mapping buttons 83
  - resizing 82
  - scaling 82
  - supported types 81
- importing
  - forms 22
- includenamespace
  - submissions (XML Model) 179
- insert
  - action 156
- insert action 156
- inserting
  - creating table row button (XForms) 156
  - guides on a ruler 40
  - item on page 37
- Instance view
  - overview 10
- instanceid
  - submissions (XML Model) 179

- interface
  - overview 5
- invisible
  - items 48
- items
  - adding context-sensitive help 54
  - aligning 43
  - changing build order 42
  - copying 48
  - copying property setting 51
  - creating error messages 55
  - cutting 48
  - displaying items to user 40
  - expanding/resizing 46
  - inserting onto a page 37
  - Palette types 29
  - pasting 48
  - resizing 41
  - selecting 41
  - setting properties 51
  - showing advanced properties 53
  - showing/hiding 40
  - snapping to 39
  - specifying type of data accepted 59
  - user object library 182
  - XForms 138

## J

- JAR
  - custom functions 101
  - embedding into form 102
  - install location 102
- Java language
  - custom functions 101
- JPEG 81

## K

- keep filtering 70, 122

## L

- label (XFDL)
  - formatting 59
- labels (XForms)
  - binding 140
  - creating 139, 140
- layout
  - bounding box 37
  - constraints defined 61
  - converting XFDL to XForms 49
  - copying items 48
  - creating guides 40
  - cutting items 48
  - deleting guides 40
  - deleting items 48
  - formatting input 59
  - grid color 38
  - grid size 38
  - layout overview 29
  - mandatory fields 62
  - moving guides 40
  - Palette item types 29
  - planning formulas 85
  - presentation properties 63

- layout (*continued*)
  - reformatting users data 63
  - rulers and guides overview 39
  - showing grid 38
  - showing/hiding ruler 39
  - snapping to grid 39
  - snapping to item 39
  - specifying accepted data types 59
  - using grids 38
  - using spacers 45
  - visibility 48

- libraries
  - calling custom functions 101
  - palette 7
  - pinning 8
  - system libraries 101
  - Viewer functions 101

- link buttons 71

- linking

- resources to projects 14

- List (Select)

- creating 146
- reordering list items 147

- List (Select1)

- adding list items 146
- creating 146
- reordering list items 147

- list choices (XForms)

- defining list items 144
- displaying from a node 142
- storing 144

- list items

- deleting 147

- list items (XForms)

- deleting 147
- displaying 143
- reordering 147

- lists

- create box lists 79
- creating popup lists 77
- formatting 59
- types 75
- when to use (XFDL) 75

- lists (XForms)

- binding to a node set 143
- creating, overview 142
- designing tips 141
- overview 141
- setting the selected value 144

- logical operators

- if/then/else 90

## M

- Make Active
  - case 150
- mandatory fields
  - setting 62
- mandatory signatures 125
- maximizing
  - views 8
- mediatype
  - submissions (XML Model) 179
- Microsoft CryptoAPI signatures 110, 113
- minimizing
  - views 8

- model (XForms)
  - adding submissions 165
- model binds (XForms)
  - overview 162
- model item properties
  - overview 163
- moving
  - guides 40
  - items 41
- multi-line fields (XForms)
  - creating 141

## N

- namespace (XML Model)
  - changing 177
  - converting attributes (XML Model) 177
- naming
  - data instances (XForms) 134
  - data instances (XML Model) 175
  - submissions (XForms) 165
  - XForms Model 131
- navigating
  - between pages 28
  - closing project 13
  - in projects 13
  - opening project 13
- Navigator view
  - overview 9
- Netscape signatures 110, 116
- network
  - share form templates 20
- no-lock signatures 120
- node (XML Model)
  - binding 178
- nodes (XForms)
  - overview 134
- nodeset
  - binding (XForms) 162
- numbers
  - formatting types 60

## O

- object library
  - users create 182
- objects
  - exporting 183
- omit filtering 70, 122
- opening
  - forms 17
  - older form 17
  - projects 13
- operators
  - formula operations 96
  - if/then/else 90
- outline view
  - build order 42
  - selecting items 41
- overview
  - bind (XForms) 162
  - bind property (XForms binding) 162
  - creating an XML Model 174
  - creating XForms lists 142
  - data submission (XForms) 165

- overview (*continued*)
  - editors 6
  - Enclosures view 10
  - help messages (XForms) 159
  - Instance view 10
  - interface 5
  - interface views 8
  - model bind (XForms) 162
  - model item properties 163
  - Navigator view 9
  - nodeset property (XForms binding) 161
  - Outline view 9
  - Problems view 9
  - Properties view 9
  - submission rules (XForms) 165
  - submissions (XForms) 165
  - submissions (XML Model) 178
  - tables (XForms) 153
  - XForms view 10
  - XML Model 173
  - XML Model components 173
  - XML Model Instance view 11
  - XML Model view 11

## P

- pages
  - adding images 81
  - design tips 25
  - hiding page size 40
  - inserting item 37
  - planning info 25
  - setting the size 26
  - showing page size 20
  - zooming 40
- Palette 6
  - add user object library 182
  - bounding box around item 37
  - changing layout 181
  - choice items 75
  - customizing 181
  - hiding 8
  - icons, large 181
  - items types 29
  - libraries overview 7
  - pinning open libraries 8
  - showing 8
  - spacer item 45
- parent nodes (XForms) 134
- password fields
  - creating 141
- pasting
  - items 48
- perspectives
  - creating user defined 182
  - Designer 5
  - selecting 5
- pixels
  - sizing/expanding items 46
- planning
  - custom functions 101
  - form workflow 4
  - items on a form 3
  - visibility 48
  - when to use formulas 85
- PNG 81

- popup lists
  - creating 77
- posting
  - submission type (XForms) 168
- preferences
  - form templates 20
- presentation
  - reformatting data 63
- Preview 7
  - overview 6
- printing
  - creating print buttons 71
- Problems view
  - overview 9
- projects
  - close/open 13
  - closing 13
  - creating 13
  - deleting 15
  - linking file or folder 14
  - navigator 13
  - opening 13
  - overview 13
  - renaming 14
- Properties
  - copy settings 51
  - help message types 54
  - resetting to default 51
  - setting global properties 19
  - setting items 51
  - show/hide categories 53
  - showing advanced properties 53
  - submissions (XML Model) 179
  - view overview 9
  - XForms label 139
- put
  - submission type (XForms) 168

## R

- radio button (XFDL)
  - creating 77
  - how they work 77
  - when to use 75
- Readonly property (XForms) 163
- recovering
  - form 21
- ref
  - binding 161
  - submission (XML Model) 179
- ref property
  - overview (XForms binding) 161
- reference
  - formula types 97
- relative align
  - defined 43
- relative expand
  - defined 46
- relative positioning
  - bounding box 37
- Relevant property (XForms) 163
- removing
  - elements (XForms) 136
  - items from form 48
  - relative positioning 46
- renaming
  - attributes (XML Model) 177

- renaming (*continued*)
  - elements (XForms) 136
  - forms 19
  - instance elements (XML Model) 176
  - projects 14
- reordering
  - list items in a list (XForms) 147
- Repeat 151, 153
- repeat item
  - binding 154
- replace buttons 71
- replacing
  - submitted data 169
- Required property (XForms) 163
- resizing
  - expanding items 46
  - images 82
  - items 41
- resource
  - linking to projects 14
- returning
  - submitted data 169
- root elements (XForms)
  - overview 134
- root nodes (XForms) 134
- routing
  - forms 68
- RSA signatures 109, 112
- rulers
  - inserting guides 40
  - overview 39
  - showing/hiding 39

## S

- sample
  - display current date 100
  - formula 98
- save buttons 71
- saving
  - forms 18
  - new file name 19
- scaling images
  - imagemode property 82
- schema based form design (XForms) 133
- screen reader
  - adding help messages 55
- select
  - XForms 141
- Select (List)
  - deleting list items 147
  - setting the appearance 147
- Select1 (List)
  - deleting list items 147
- selecting
  - cases 150
  - items 41
  - Palette layout 181
  - perspectives 5
- setindex
  - action 157
- setindex action 156
- setting
  - appearance XForms lists 147
  - bind properties (XForms) 162
  - data to be submitted using bind (XForms) 167

- setting (*continued*)
    - data to be submitted using ref (XForms) 167
    - default case 150
    - Select (List) appearance 147
    - submission rules (XML Model) 178
    - submission type (XForms) 168
    - target URL submission (XForms) 168
    - value from a list (XForms) 144
    - XForms label properties 139
  - shortcut keys
    - customizing 181
  - Show Advanced Properties 9
  - showing
    - items 40
    - page size 40
    - Palette 8
    - ruler 39
    - tab order 42
    - views 8
  - sibling elements (XForms)
    - adding 136
    - overview 134
  - sibling nodes (XForms) 134
  - sid
    - toolbar 52
  - Signature Pad signatures 117
  - signatures 107
    - approval 120
    - Authenticated Clickwrap 112
    - Authenticated Clickwrap button 115
    - Clickwrap 111, 114
    - digital 109
    - engines 109
    - Entrust 110, 116
    - filtering 121
    - Generic RSA 109, 112
    - mandatory 125
    - Microsoft CryptoAPI 110, 113
    - Netscape 110, 116
    - no-lock 120
    - properties 125
    - Signature Pad 110, 117
    - Silanis 111, 119
    - types 109
  - signing a form 108
  - Silanis signatures 111, 119
  - single line fields
    - creating 140
  - sizing
    - items in pixels 46
    - pages 26
  - Smartfill
    - overview 169
    - XForms 169
  - snapping
    - to grid 39
    - to item 39
  - source panel
    - overview 6
  - spacer
    - Palette item 45
  - standalone
    - labels (XForms) 140
  - starting up 3
  - storing
    - list choice (XForms) 144
  - string
    - formatting types 60
  - submission bind property (XForms)
    - submitting data 167
  - submission button
    - XML Model 179
  - submission button (XForms)
    - creating 169
  - submission ref property (XForms)
    - submitting data 167
  - submission rules (XML Model)
    - deleting 179
    - setting 178
  - submission type (XForms)
    - setting 168
  - submissions (XFDL)
    - filtering 70
  - submissions (XForms)
    - adding to a data instance 165
    - adding to a model 165
    - naming 165
    - overview 165
    - setting target URL 168
    - specifying binds 165
    - specifying nodes 165
  - submissions (XML Model)
    - action 179
    - adding 178
    - id 179
    - includenamespace 179
    - instanceid 179
    - mediatype 179
    - overview 178
    - properties 179
    - ref 179
  - submit button (XForms) 169
  - submit buttons (XFDL) 68
  - submitted data
    - displaying 170
  - submitting data (XForms)
    - using bind property 167
    - using ref property 167
  - sum
    - formula 88
  - SunRaster 81
  - support 130
  - Switch
    - adding 149
  - switching
    - between cases 150
    - perspectives 5
  - system libraries
    - shipped functions 101
- ## T
- tab order
    - changing build order 42
    - showing form flow 42
    - tab order overview 42
    - view menu 42
  - tables
    - XForms 151
  - tables (XForms) 151
    - creating 151
    - overview of manual setup 153
  - template
    - creating row (XForms) 155
    - images 83
    - object items 182
  - templates
    - description 20
    - preview image 20
    - using form templates 20
  - testing your form 7
  - text values (XForms)
    - overview 134
  - time
    - formatting types 60
  - toggle action 151
  - toggle between cases 150, 151
  - toggle case button 150, 151
    - creating 150
  - toolbar
    - build order 52
    - color 52
  - transmissions
    - HTTP or E-mail forms 68
  - transmitformat 179
  - Trigger (Button)
    - toggle between cases 150
  - Type property (XForms) 163
- ## U
- undo
    - moving item 41
  - upgrading
    - forms 18
  - uploading
    - attachments 105
  - URL
    - setting target (XForms) 168
  - user interface
    - Designer overview 5
    - XForms 138
- ## V
- viewing your form 7
  - views
    - close 8
    - Enclosures 10
    - Instance 10
    - maximize 8
    - minimize 8
    - Navigator 9
    - Outline 9
    - overview 8
    - Problems 9
    - Properties 9
    - show 8
    - XForms 10
    - XML Model 11
    - XML Model Instance 11
  - visibility
    - display items in Viewer 48
    - print items 48
  - visually impaired
    - adding audio help 55



## W

warnings

Problems view 9

Web services

adding to form 127

overview 127

wizard

Compute Wizard 86

WSDL

adding to form 127

overview 127

WSDL messages

creating data instances 134

deleting 127

## X

XFDL

changing Format constraints 61

compress format 68

converting to XForms 49

converting to XForms items 160

Format constraints defined 61

presentation properties defined 63

XForms

adding support to new form 130

adding to form 130

conditional items 148

converting XFDL items 49

creating 142

creating a table by wizard 151

creating empty data instance 133

creating tables manually 151

data instances, creating 132

data instances, creating from

document 132

data instances, creating from WSDL

message 134

data instances, overview 131

deleting WSDL file 127

fields, overview 140

items 138

label properties 139

lists 141

lists, overview 142

overview 129

schema based form design 133

Smartfill 169

support 130

table, adding items 155

user interface, overview 138

visibility 48

when to use 129

XForms Model

id 131

naming 131

overview 130

XForms view

overview 10

XML Model

adding 174

binding 177

Instance view overview 11

overview 173

submission button 179

view overview 11

XML Model views

displaying 174

## Z

zooming

in/out of the page 40









Program Number: 5724-N07

Printed in USA

S325-2589-00

