Novell exteNd Director

5.2

www.novell.com

PAGEFLOW AND FORM GUIDE

Novell®

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.

www.novell.com

exteNd Director*Pageflow and Form Guide*
June 2004

Online Documentation: To access the online documemntation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

ConsoleOne is a registered trademark of Novell, Inc.
eDirectory is a trademark of Novell, Inc.
GroupWise is a registered trademark of Novell, Inc.
exteNd is a trademark of Novell, Inc.
exteNd Composer is a trademark of Novell, Inc.
exteNd Director is a trademark of Novell, Inc.
iChain is a registered trademark of Novell, Inc.
jBroker is a trademark of Novell, Inc.
NetWare is a registered trademark of Novell, Inc.
Novell is a registered trademark of Novell, Inc.
Novell eGuide is a trademark of Novell, Inc.

SilverStream Trademarks

SilverStream is a registered trademark of SilverStream Software, LLC.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Third-Party Software Legal Notices

The Apache Software License, Version 1.1

Copyright (c) 2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

JDOM.JAR

Copyright (C) 2000-2002 Brett McLaughlin & Jason Hunter. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (http://www.jdom.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun

Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun Logo Sun, the Sun logo, Sun Microsystems, JavaBeans, Enterprise JavaBeans, JavaServer

Pages, Java Naming and Directory Interface, JDK, JDBC, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, SunWorkShop, XView, Java WorkShop, the Java Coffee Cup logo, Visual Java, and NetBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Indiana University Extreme! Lab Software License

Version 1.1.1

Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. 5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Phaos

This Software is derived in part from the SSLavaTM Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved. Customer is prohibited from accessing the functionality of the Phaos software.

W3C

W3C® SOFTWARE NOTICE AND LICENSE

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications: 1.The full text of this NOTICE in a location viewable to users of the redistributed or derivative work. 2.Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code. 3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Contents

About This Book

Purpose

This book introduces the basic concepts, architecture, and tools of the Novell® exteNd Director™ Pageflow subsystem.

Audience

This book is for anyone who needs to design pageflow processes or understand the features of the Pageflow subsystem.

Prerequisites

This book assumes knowledge of Java programming and familiarity with HTML and XML.

Additional documentation

📖 For the complete set of Novell exteNd Director documentation, see the Novell Documentation Web Site (http://www.novell.com/documentation/).

Concepts

Provides an overview of pageflow concepts

1 About Pageflows and XForms

This chapter provides an introduction to the use of pageflows and XForms in exteNd Director applications. It includes these topics:

What is a pageflow?

A *pageflow* defines the flow of control for a set of pages that execute within a single portlet. Each page presents a set of controls that allow for user interaction. For example, the pages in a flow might provide a way for the user to display stock quotes or weather forecasts, or access corporate data such as employee information.

The pages within a flow can use either of the following technologies to define the user presentation:

- HTML
- XHTML (XForms)

In addition to presenting pages for user interaction, pageflows can perform background processing tasks. For example, a pageflow might invoke a Web Service, access a database, or simply execute code written in Java.

Wizards To facilitate database access, exteNd Director provides tools for creating pageflows that give the user a way to find, display, and modify records in a database. exteNd Director provides the **Database Pageflow Wizard** to help you create these pageflows. The Database Pageflow Wizard generates a set of forms (XHTML pages that use XForms technology) as well as one or more pageflows that tie the forms together into an integrated user interface.

exteNd Director also provides tools for creating pageflows that invoke Web Services. To help you create these kinds of flows, exteNd Director provides the **Web Service Pageflow Wizard**. exteNd Director also provides the **Composer Pageflow Wizard** to help you take advantage of exteNd Composer™ services.

Pageflow components

The core of a pageflow application is the *process descriptor*, an XML file that you create visually using the Pageflow Modeler. A *pageflow process*, which is the visual representation of the process descriptor, is a branching series of activities and links that models a set of user interactions within a portlet. These are the key parts of a pageflow process:

Process component	Description
Activity	An object that represents a task. An activity can present information to the user and respond to user interactions, or perform background functions that are not visible to the user. 📖 For more information on activities, see Chapter 3, "Working with Activities".
Link	An object that represents a path in the routing logic of the flow. A link points to an activity. Links are what tie the activities in a pageflow together. A link is a single logical path between two activities. A link can also move data between activities. An activity can have multiple source (incoming) links and multiple destination (outgoing) links. 📖 For more information on links, see Chapter 4, "Working with Links".

Pageflow data

A typical pageflow process includes data that is manipulated by pageflow users or program logic. To access data in a pageflow, you use **scoped paths**. exteNd Director includes a group of predefined scoped paths that are available from the Pageflow Modeler and the Workflow Modeler and through the Scoped Path API.

📖 For more information on scoped paths, see "Using scoped paths in a pageflow" on page 21.

Pageflow engine

The *pageflow engine* is responsible for executing and managing workflow processes. It uses the process descriptor created with the Pageflow Modeler to:

- Create new process instances
- Start and stop activities
- Execute the link to the next activity

Workflow and pageflow

It is helpful to understand pageflow by contrasting it with the exteNd Director workflow model. Although workflow and pageflow share similar mechanics, pageflow has some distinguishing features, as shown here:

Workflow	Pageflow
Process-based	**Session-based**
A workflow is a linear business process that might span several days, or even weeks.	A pageflow is essentially open-ended, and usually shorter (less than an hour) in duration.

Workflow	Pageflow
Definite starting point Some specific event or condition triggers a process instance. This can be a system-generated event, like a monthly notification, or user-generated, as in a telephone sales transaction.	**Designed entry point** The entry point is determined by the pageflow itself and is typically not triggered by an external event, unless the pageflow is being used in a workflow.
Multiple users A workflow assumes multiple users are performing discrete tasks at each activity.	**Single user** A pageflow application is driven by a single user.
Persistent data Instance data must be stored outside the session so that workitems can be passed to subsequent activities.	**Session data** A pageflow relies on instance data stored in the current session, although the application can access persistent data.
Definite ending point Some specific event or condition ends the process instance. In a sales order transaction, the business process cannot conclude until the customer verifies receipt.	**User-controlled exit point** The user chooses when to end the session, although some specific action could trigger a workflow.

NOTE: Although workflow and pageflow are different types of applications, the Workflow subsystem provides facilities for integrating them. You can embed a pageflow in a workflow using the Pageflow activity in the Workflow Modeler. You can also start a workflow process from a pageflow using a Java activity.

 For details on embedding on a pageflow in a workflow, see the chapter on working with activities in the *Workflow Guide*. For details on starting a workflow process from a pageflow, see “Example: Starting a workflow process” on page 165.

What is XForms technology?

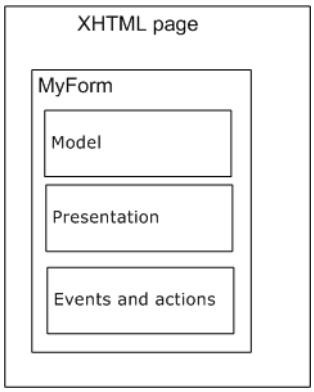
XForms provide a robust, standards-based way to define Web forms. The advantages of the XForms standard include:

- Separate data, logic, and presentation modules
- A powerful event model (so that you don’t have to use a lot of scripting for client-side validation or calculations)
- A way to process data in XML formats

XForms cannot run as standalone applications. They are designed to run as components within a host language like XHTML. In the Novell implementation, XForms run within the context of a pageflow application.

XForms structure

A single XForms document includes the sections shown in the following diagram:



Section	Description
Model	An element that defines the structure of the XML data used by the form. The Model defines the data that is: ◆ Displayed or entered by the user ◆ Submitted to the server ◆ Used for temporary calculations The Model element might also include rules for validation, constraints, and calculations.
Presentation	The XForms 1.0 Specification defines a set of abstract controls you use to define the user interface. You can bind the controls to data elements defined in the Model.
Events and actions	Defines the form's processing logic. XForms Actions provide built-in logic to run in response to events. Events and actions can be programmed on both Model and presentation elements.

📖 For more information on using the Form Designer to define the Model, presentation, or events for an XForms document, see Chapter 6, "Form Designer".

Form data

In XForms, the data is defined in the Model element. The Model can contain one or more instance elements.

At design time, you provide the Form Wizard with an XML schema or instance document that defines the structure of the instance data. Once an initial form is created based on this structure, you'll use the Form Designer's Model Editor to define additional information about the data such as constraints, calculations, validations, and so on. In addition, you'll define how the runtime data is:

- Supplied—since the form runs in the context of a pageflow, you'll specify the circumstances when the data is supplied by the pageflow (if the instance data is inline).
- Submitted—you'll specify what part of the data to submit, and the other submission details such as: the method (PUT, POST, GET) and the encoding.

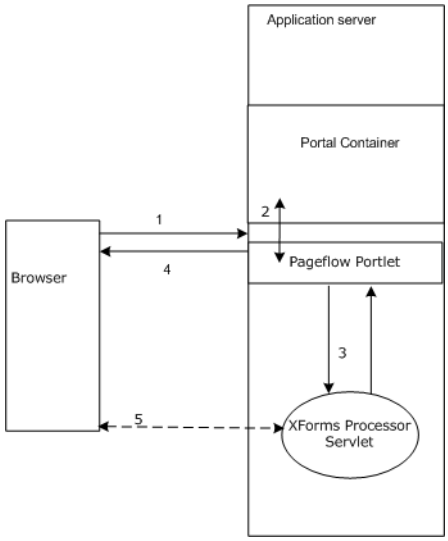
XForms processing

XForms can be processed via a browser plug-in or a server-side *XForms processor.*

At startup the exteNd Director portal checks to see if a browser plug-in is installed. If not, XForms processing defaults to the Novell exteNd Director server-side processor. The server-side processor is responsible for:

- Generating an in-memory model of the form
- Generating HTML and Javascript so that a browser can render an XForm.
- Performing the data validation, calculations, constraints and so on defined on the form.
- Responding to XML events and XForms actions that occur on the form.

The following diagram illustrates the interaction flow of the Novell exteNd Director server-side XForms processor and a device such as a browser.



1 The browser makes a request for an XForms document—running in a pageflow.

2 The pageflow portlet receives the request (via the portal container), renders it, and passes it to the XForms server-side processor.

3 If a browser plug-in does not exist, the XForms exteNd Director server-side processor:

- Builds an in-memory model of the form.
- Renders the form as HTML and Javascript (to render the XHTML in a format that a browser can understand).
- The XForms processor returns the markup fragment to the pageflow portlet.

4 The Portal returns the markup to the browser.

5 The browser establishes a network connection to the XForms server-side processor so all subsequent communications within the form (for example, event processing) are direct with the XForms server-side processor, bypassing the Portal container.

About the Pageflow Modeler

Integrated into the development environment is a complete graphing package called the *Pageflow Modeler* that allows you to quickly and visually create a pageflow process. The Pageflow Modeler allows you to:

- Graphically lay out, annotate, and format your pageflow
- Create, change, and delete activities and links
- Set activity and link properties
- Use the Scoped Path Navigator to associate activities and links with data

When you save a pageflow, the Pageflow Modeler translates your document into an XML-based file called a *process definition*. The process definition saves the layout and format of your pageflow and translates the flow logic into a program the pageflow engine can read and execute.

The Pageflow Modeler also saves a *portlet fragment deployment descriptor* that maps your pageflow to a pageflow runner. The pageflow runner is a Java class that is implemented as a portlet.

About the Form Designer

The exteNd Director Form Designer provides an environment for developing XForms 1.0-compliant Web forms. The Form Designer is a graphical development tool that allows you to quickly create XForms components for use in pageflow applications.

2 Working with Pageflows

This chapter introduces pageflow processes and provides several examples of complete pageflows. It includes these topics:

- About the pageflow process
- About the pageflow portlet descriptor
- Using scoped paths in a pageflow
- Pageflows and portlet runtime behavior
- Pageflow logging
- Examples

About the pageflow process

The core of a pageflow application is the *process descriptor*, an XML file that you create visually using the Pageflow Modeler. A *pageflow process*, which is the visual representation of the process descriptor, is a branching series of activities and links that models a set of user interactions within a portlet.

The pageflow process descriptor is placed in the **pageflow-process** folder within the resource set.

Process object Each pageflow process contains a global object called the *process object* that defines some general settings for the process as a whole.

📖 For details on setting the properties associated with the process, see "Process properties" on page 67.

Subflows (flows within flows) You can include a pageflow process within another pageflow process. When you do this, the embedded flow is included within the containing flow.

About the pageflow portlet descriptor

When you save a pageflow, the Pageflow Modeler saves a *portlet fragment deployment descriptor* in the resource set that maps your pageflow to a *pageflow runner*. The pageflow runner is a Java class that is implemented as a portlet. exteNd Director ships with a prepackaged pageflow runner (com.novell.afw.portal.portlet.pf.pageFlowRunner). This class can run any pageflow. You can write your own pageflow runner, but this is not necessary.

📖 For more information about the portlet fragment deployment descriptor, see the section on the portlet fragment deployment descriptor in the *Portal Guide*.

Each pageflow has a separate descriptor Although all pageflows use the same portlet runner class, each pageflow has its own descriptor. The descriptor has a unique name that distinguishes it from other portlet descriptors. By default, the name given to the descriptor is the same as the name given to the pageflow. Therefore, when you save a pageflow in the Pageflow Modeler, you get two XML descriptors with the same name. These descriptors are placed in different folders within the resource set:

- The pageflow **process descriptor** is placed in the **pageflow-process** folder
- The pageflow **portlet descriptor** is placed in the **portal-portlet** folder

To display a pageflow on a portal page, the user must select the portlet that runs the pageflow (not the pageflow itself).

What's inside the descriptor Each portlet descriptor created for a pageflow specifies distinct initialization parameters, settings, and preferences. One of the initialization parameters, called **PF_ID**, specifies the name of the pageflow process to run:

```
<init-param>
  <description>Pageflow ID</description>
  <name>PF_ID</name>
  <value>MyPageflow</value>
</init-param>
```

Sample pageflow portlet descriptor Here is an example of a portlet descriptor that was generated for a pageflow called MyPageflow:

```
<portlet xmlns="http://www.novell.com/xml/ns/portlet-fragment"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <description>MyPageflow</description>
   <portlet-name>MyPageflow</portlet-name>
   <display-name>MyPageflow</display-name>
   <portlet-class>com.novell.afw.portal.portlet.pf.pageFlowRunner
   </portlet-class>
   <init-param>
      <description>Pageflow ID</description>
      <name>PF_ID</name>
      <value>MyPageflow</value>
   </init-param>
   <expiration-cache>0</expiration-cache>
   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
   </supports>
   <portlet-info>
      <title>MyPageflow</title>
      <short-title>MyPageflow</short-title>
      <keywords>MyPageflow</keywords>
   </portlet-info>
   <portlet-preferences>
      <preference>
         <name>width</name>
         <value>400px</value>
         <read-only>false</read-only>
         <data-type>String</data-type>
         <required>false</required>
         <multi-valued>false</multi-valued>
      </preference>
      <preference>
         <name>height</name>
         <value>400px</value>
         <read-only>false</read-only>
         <data-type>String</data-type>
         <required>false</required>
         <multi-valued>false</multi-valued>
      </preference>
      <preference>
```

```
            <name>Restrict Portlet Size</name>
            <value>false</value>
            <read-only>false</read-only>
            <data-type>Boolean</data-type>
            <required>false</required>
            <multi-valued>false</multi-valued>
         </preference>
      </portlet-preferences>
      <enable-title-bar>1</enable-title-bar>
      <supported-option>Restart</supported-option>
      <supported-option>edit</supported-option>
      <preview-image></preview-image>
      <auto-register enabled="true">
         <registration-id>MyPageflow</registration-id>
      </auto-register>
   </portlet>
```

Pageflow portlet preferences The portlet descriptor for a pageflow includes three preferences that can be used to control the runtime display of the portlet:

Preference	Description
Restrict Portlet Size	Controls whether tags are placed around the result of the pageflow request. This is useful for flows that are very wide or tall. The tags will be added to the HTML only if Restrict Portlet Size is set to **true** and the portlet is not in a maximized state. **NOTE:** When the Restrict Portlet Size preference is set to **true**, the pageflow runner portlet automatically sets the content type to **text/html**. The default value for Restrict Portlet Size is **false**.
width	Specifies the width (in pixels) when the Restrict Portlet Size preference is set to **true**. This preference is ignored when Restrict Portlet Size is set to **false**. The default value for width is **400px**.
height	Specifies the height (in pixels) when the Restrict Portlet Size preference is set to **true**. This preference is ignored when Restrict Portlet Size is set to **false**. The default value for height is **400px**.

Using scoped paths in a pageflow

Scoped paths allow you to access different kinds of data from your pageflow process. exteNd Director includes a group of predefined scoped paths that are available from the Pageflow Modeler and through the Scoped Path API.

Data can be in either a nonpersistent or a persistent state. Data is nonpersistent if it is available to a single user for a single application session. Data is persistent if it is available to multiple users and, potentially, other applications. In the context of scoped paths, a path is the physical location of the data within a specified scope.

Scoped paths in the Pageflow Modeler

In the Pageflow Modeler, you use scoped paths in several ways:

- To bind an activity to an object (such as a rule, Java class, Web Service, HTML document, XHTML form, or other external pageflow)
- To specify the instance data for an activity
- To copy data from one location to another

Scopes you can use within a pageflow

All of the predefined scopes are available within a pageflow.

📖 For complete details on using the predefined scopes, see the chapter on scoped paths in *Developing exteNd Director Applications*.

These are some examples of scoped paths you might use within a pageflow.

Scoped path	Possible runtime value
/Request/prop/User-Agent	Mozilla 6.0 (MSIE 6.0)
/String/true	true
/ResourceSet/html/myHTML.html	The contents of the myHTML.html file.
/ResourceSet/document-template/employees.xml/emp_list/employees[@id='12345']/firstName	The first name string of the employee whose ID is **12345**. This value is located in the employees.xml file within the resource set.
/Flow/document/InstanceData	The contents of an XML document stored in the document folder of the Flow scope.

Restrictions Although you can use any of the predefined scopes within a pageflow, some restrictions apply to the Request and Response scopes. For more information, see "Pageflows and portlet runtime behavior" on page 22.

Pageflows and portlet runtime behavior

This section discusses some of the ways in which portlet runtime behavior can change the behavior of a pageflow.

Scoped paths and portlet runtime context

At each point in the execution of a pageflow, the runtime behavior of the pageflow runner portlet has a direct effect on what kind of information is available to scoped paths. Therefore, when you're deciding which scoped paths to use in a pageflow, you need to be aware of the underlying portlet runtime context, particularly when using the Request and Response scopes.

Restrictions that apply to the Request and Response scopes When working with the Request and Response scopes, you need to be aware of these restrictions:

- The **api** option on the **Request** scope is available only on render requests (in other words, during the execution of the render() method of the pageflow runner portlet).
- The **render** option on the **Response** scope is also available only on render requests (within the execution of the render() method of the pageflow runner portlet).

All other Request and Response options are available on either a render (within the execution of the render() method of the pageflow runner portlet) or an action (within the execution of the processAction() method of the pageflow runner portlet).

Determining which portlet runtime context is available The following rules determine which context applies at each stage of processing within a pageflow:

- **When a pageflow is first loaded**, the pageflow runner portlet executes the render() method. If the first activity following the Mode activity is a presentation activity (HTML, Form, JSP, or Servlet), the runner performs any Copy Before operations specified for the activity and then executes the presentation activity itself.

  If there are any system activities (such as a Web Service activity, Composer activity, or Java activity), the runner portlet executes these activities in order. Any Copy Before and Copy After operations specified for these activities are also performed.

  All Copy Before and Copy After operations are therefore performed within the render() phase from the time the pageflow is first initiated until the first presentation activity is executed.

- **If an action occurs on the first presentation activity within the flow**, any Copy After operations associated with this activity are performed. The Copy After processing occurs within the processAction() method. However, if a rerender occurs (because the user clicks on a render URL or hits the browser refresh), the Copy After is performed only if the activity is preceded by a CheckPoint activity. If it is not preceded by a CheckPoint activity, the Copy After operation is not performed. The reason for this is that the Copy After is performed only when the engine moves to another activity (as it does in the case of the CheckPoint activity).

  In the event of a rerender, all Copy After processing occurs within the render() method.

  NOTE: Different contexts are being used under the covers for the processAction() and render() phases, so it is important to be aware of which context is available.

- **When system activities are executed within a pageflow**, all Copy Before and Copy After processing occurs in the phase in which they run. If a system activity follows a CheckPoint, the Copy Before and Copy After operations will be executed within the render() method. If the activity does not occur after a CheckPoint, then the Copy Before and Copy After operations all occur in the processAction() method.

  NOTE: All system activities executed before the first presentation activity are executed within the render() method, regardless of whether a CheckPoint is present.

Pageflow performance and portlet caching

The caching behavior of the pageflow runner portlet has a direct effect on the performance of a pageflow.

You can define an **expiration cache** in the portlet descriptor for a pageflow:

- A value of **0** disables caching for the portlet. When you specify 0, the portlet content is never cached.
- A value of **-1** means the cached content never expires for that portlet. When you specify -1, the cached content is used indefinitely.
- Any other **positive value** defines an expiration cache whose content expires after the number of seconds specified by the value. A positive value indicates the number of seconds to cache before allowing render calls.

If you want to minimize the number of render operations associated with a pageflow, you may want to set the expiration cache to a positive number.

Here's an example that shows how you might set the expiration cache:

```
<portlet xmlns="http://www.novell.com/xml/ns/portlet-fragment"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <description>MyPageflow</description>
   <portlet-name>MyPageflow</portlet-name>
   <display-name>MyPageflow</display-name>
   <portlet-class>com.novell.afw.portal.portlet.pf.pageFlowRunner</portlet-class>
<init-param>
      <description>Pageflow ID</description>
      <name>PF_ID</name>
      <value>tdbphonelistPageflow</value>
</init-param>
<expiration-cache>1000</expiration-cache>
...
</portlet>
```

Pageflow logging

The following logs are provided to help you gather information about the runtime behavior of a pageflow:

Log	Description
PageFlowLog	Logs information that is common to all pageflows.
PageFlowFormLog	Logs information related to XForms processing associated with pageflows.

To modify the settings associated with these logs, you can edit the **config.xml** file for the Pageflow subsystem.

For example, you might want to increase the logging level for these logs from 3 to 5 to get more information about pageflows running in your application:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties PUBLIC "-//SilverStream Software, LLC//DTD Framework Config
XML 3.0//EN" "framework-config_3_0.dtd">
 <properties>
    <property>
        <key>PageflowService/engine-id</key>
        <value>engine-id</value>
    </property>
    ...
    <property>
        <key>PageFlowLog.LoggingLevel</key>
        <value>5</value>
    </property>
    <property>
        <key>PageFlowLog.LogFieldSeparator</key>
        <value>|</value>
    </property>
    <property>
        <key>PageFlowLog.LoggingProvider</key>
        <value>com.sssw.fw.log.EboStandardOutLoggingProvider</value>
    </property>
    <property>
        <key>PageFlowFormLog.LoggingLevel</key>
        <value>5</value>
    </property>
    <property>
        <key>PageFlowFormLog.LogFieldSeparator</key>
        <value>|</value>
    </property>
    <property>
        <key>PageFlowFormLog.LoggingProvider</key>
        <value>com.sssw.fw.log.EboStandardOutLoggingProvider</value>
    </property>
    ...
</properties>
```

NOTE: You can also change the logging levels in the Director Administration Console.

Examples

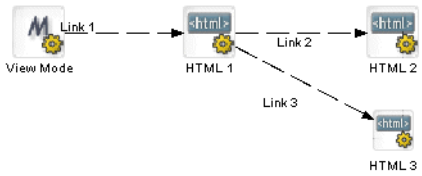
This section presents some examples that illustrate basic pageflow concepts.

Example 1: Simple flow with HTML activities

This is a simple pageflow that shows the required Mode Activity, followed by two HTML activities connected by simple links. HTML activities are presentation activities that display static pages to the user:

Example 2: Flow with link expressions

This example shows how link expressions can be used to control the navigation paths within a pageflow. The expressions for Link 2 and Link 3 determine which button the user clicked on HTML 1. Depending on which button was clicked, the user sees HTML 2 or HTML 3:



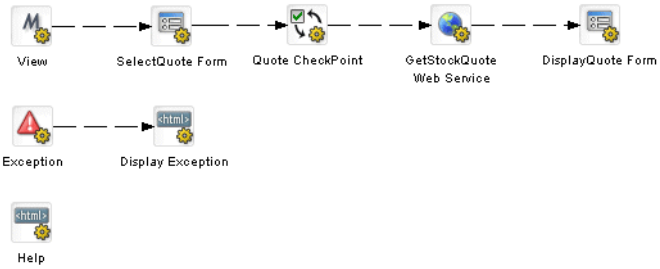
Example 3: Flow with a system activity

The following flow shows the use of the XSL activity, which is a system activity. XSL activity 1 transforms some XML instance data. The resulting HTML output is then used as input to HTML 1.

Since the XSL activity is a system activity, it is not visible to the user. When this flow is initiated, the user sees only the HTML page:

Example 4: Web Service and XHTML forms

This flow contains a Web Service activity, another example of a system activity. This activity invokes a Web Service that gets a stock quote. Before the Web Service is invoked, the user specifies the stock symbol on an XHTML form associated with a Form activity. When the Web Service has finished processing, a second Form activity displays the quote results. To handle browser refreshes, this flow includes a CheckPoint activity. Whenever a refresh occurs, the flow returns to the Quote CheckPoint activity and continues from that point forward:



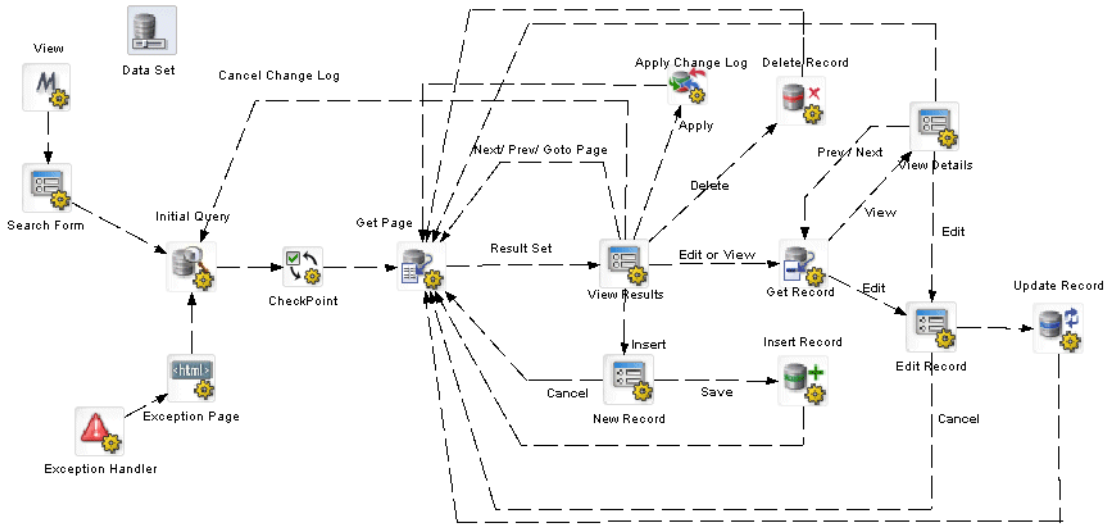
An Exception activity has also been added to this flow to handle exceptions that may occur at runtime. For example, if the Web Service being called is unavailable for some reason, the Exception activity takes control. This activity forwards processing to an HTML activity called Display Exception that displays a helpful message to the user.

This flow also shows the use of *smart linking*. Smart linking reduces clutter within a pageflow when many activities need to link to a common activity. In this example, the Help HTML activity is accessed by means of a dynamic link generated at runtime.

Example 5: Database pageflow

This example shows a typical database pageflow used to access a single table. It includes several database activities and forms that allow the user to enter search criteria and view search results. Also it includes activities and forms that permit the user to perform inserts, updates, and deletes.

Every database pageflow process includes a Data Set object that provides all the information required to access the database:



This flow was generated by the Database Pageflow Wizard.

For more information on the Database Pageflow Wizard, see Chapter 7, “Database Pageflow Wizard”.

3 Working with Activities

This chapter introduces the activities you can use in a pageflow process. It includes these topics:

About pageflow activities

Categories

There are several categories of pageflow activities:

Category	Description	Activites
Presentation	Control the user interface for the flow	Form HTML XML JSP Pageflow Servlet
Database	Allow a pageflow to find, display, and modify records in a database **NOTE:** The database activities are added by the Database Pageflow Wizard. Therefore, you do not usually need to add these by hand.	Initial Query Get Page Get Record Record Insert Record Update Record Delete Apply Change Log
System	Perform background processing functions required by the flow	Mode XSL Web Service Composer Service Rule Java Finish Workflow Return
Directive	Affect the underlying processing of the flow	Exception CheckPoint

NOTE: The Database tab in the Pageflow Modeler also includes the Data Set. The Data Set is not an activity in the conventional sense. At design time, you can make changes to the definition of the Data Set, just as you would make changes to any pageflow activity. But at runtime the Data Set does not behave like an activity: it does not perform any processing or affect the flow of control as the other activities do.

📖 For more information on the Data Set, see "Working with the Data Set" on page 148.

The pageflow activities have *properties* you can set in the Pageflow Modeler. Some of these properties are common to all activities, whereas others are available only on some of the activities.

Most pageflow activities have a **primary property**. The primary property associates an object with the activity. For presentation activities, this is the item to display. For system properties, this is the object you want to execute.

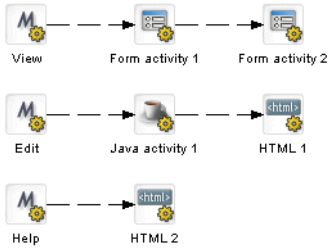
Mode activity

Description

The **Mode activity** is the starting point for a particular mode within a pageflow. The Mode activity initiates a series of activities that are performed when the process is in that mode. Each mode activity maps directly to a *portlet mode*, as defined by the Portlet specification (Java Portlet 1.0).

The Mode activity is a required activity. Each pageflow must have at least one Mode activity of type View. When you create a new pageflow process, the Pageflow Modeler automatically adds a Mode activity that has the View type.

A pageflow can have multiple Mode activities. For example, a pageflow might have Mode activities that specify what happens when the flow is in View, Edit, or Help mode:



Properties

The properties of the Mode activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Mode	Mode	Specifies the mode type. Pageflows support the following portletsmodes: ◆ View ◆ Edit ◆ Help
	Default	Indicates whether this is the default mode for the pageflow. A given pageflow can have only one Mode activity that is marked as the default.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Usage

What happens when a pageflow is initiated All pageflows start in View mode. Since the Portlet specification requires that all portlets implement this mode, View mode is required in pageflows.

What happens when the user switches modes The user can switch modes at runtime—mode switching is a runtime-only function. The pageflow runner detects mode switches by examining request information provided by the Portlet container. If a mode switch is detected, the flow is restarted using the named mode. If the mode specified cannot be found, an exception is thrown and the flow remains in its previous state. The default mode is not used during mode switching.

Form activity

Description

The **Form activity** is a presentation activity that specifies an XHTML file to display at runtime. The XHTML file defines a form for user interaction. The form uses XForms technologies to define the user interface. By providing access to the full range of XForms capabilities, the Form activity gives you an extremely powerful tool for building presentation into a pageflow process. The form displayed by a Form activity can include a variety of user controls, such as buttons, input fields, and labels. The form can also give the user a way to interact with a database or Web Service.

The exteNd Director Form Designer provides a graphical environment for developing XForms 1.0-compliant Web forms. The Form Designer lets you create and modify XHTML forms that can be displayed with the Form activity.

For details on working with the Form Designer, see Chapter 6, “Form Designer”.

Properties

The properties of the Form activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Form	XHTML	Specifies the XHTML content to display. You can create the XHTML content using the Form Designer.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Instance Data	Inbound Documents	Specifies one or more documents that are used as input to this activity. The location of each document is specified by means of a scoped path. The list of inbound documents for the activity is determined by how many inbound documents are defined in the XHTML file.
	Outbound Document Name	Specifies a document that contains output produced by this activity. The location of the document is specified by means of a scoped path.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

HTML activity

Description

The **HTML activity** is a presentation activity that specifies HTML content to display at runtime.

If the HTML content you want to display contains an HTML form, the action for the form must specify the wsrp_rewrite token, an industry-standard token defined by the WSRP specification:

```
action="wsrp_rewrite?wsrp-urlType=blockingAction/wsrp_rewrite"
```

NOTE: This version of exteNd Director does not provide full support for the WSRP protocol. But, support for many WSRP features has been implemented. For example, the wsrp_rewrite token is required in the action for a form.

XHTML content that you display in a Form activity can specify this abbreviated syntax in the action for a form:

```
<xforms:submission
   action="?verb=help"
   id="help"
   method="post"/>
```

NOTE: This syntax is not supported in HTML content.

Example 1 Here's a sample HTML page that could be displayed with an HTML activity. The HTML source for this activity has a Next button that passes a value of `next` for the parameter named `verb`:

```
<form name="form1" method="post" action="wsrp_rewrite?wsrp-
urlType=blockingAction/wsrp_rewrite">
 <p>Enter your name
    <input type="name" name="name">
    <br><br>
    <input type="submit" name="verb" value="next">
    <input type="submit" name="verb" value="help">
</form>
```

Example 2 This example shows tags that take advantage of the classes defined in the theme CSS file. By mapping your tags to classes in the CSS file, you can ensure that these tags will alter their appearance according to the display characteristics of the currently selected theme:

```
<form name="form1" method="post" action="wsrp_rewrite?wsrp-
urlType=blockingAction/wsrp_rewrite">
<br/>
    <span class="portlet-form-field-label">
        Zip
    </span>
    <input class="portlet-form-field" type="name" name="zip" value="02630"
size="20”>
    <br/><br/>
    <input type="submit" name="verb" value="Continue">
</form>
```

Properties

The properties of the HTML activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.

Property Inspector tab	Property name	Description
HTML	HTML	Specifies the HTML content to display. The HTML page referenced by an HTML activity must reside within the resource set.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

XML activity

Description

The **XML activity** is a presentation activity that specifies XML content. Pageflows can return content of type text/xml. This allows you to use portal styling to control the presentation, which is particularly useful when you want to perform transcoding for wireless applications.

A pageflow can have multiple renderable activities. When you return XML content, you must use XSL to determine which template to use for the returned XML.

To style the content for a pageflow, you need to add the proper style to the portlet fragment deployment descriptor associated with the pageflow.

Properties

The properties of the XML activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
XML	XML	Specifies the XML content to display. The location of the XML content is specified by means of a scoped path.
	Content Type	Specifies the content type to display. The content type is specified by means of a scoped path. For example, you might specify `/String/text/xml` as the content type. If you do not specify a content type, the default is `text/html`.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Pageflow activity

Description

The **Pageflow activity** includes a separate pageflow within the current flow.

The Pageflow activity performs a runtime include. The Pageflow Modeler does not perform any design-time checking to ensure that the flow specified by the Pageflow activity will actually work properly within the containing flow.

When you include a pageflow within another pageflow, the Mode activity within the included flow must have a type that matches the Mode type of the containing flow. Also the included flow must have a Finish activity. The Finish activity tells the engine when to return to the containing flow. If the included flow does not contain a Finish activity, control will never be returned to the containing flow.

When a pageflow is included within another pageflow, the Mode and Finish activities within the included flow perform any scoped path functions required and then execute a forward operation.

Properties

The properties of the Pageflow activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Pageflow	Pageflow	Indicates which pageflow to run by specifying a pageflow descriptor in the resource set.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Usage

At runtime, the Pageflow activity doesn't perform any processing itself. Instead, it just instructs the pageflow engine to insert the included pageflow within the main pageflow before the process is started.

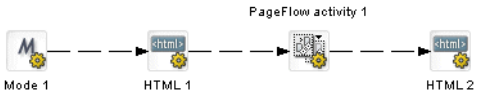
These are the basic steps the engine performs when it includes a pageflow within another pageflow:

1. Gets the pageflow to include using a scoped path.
2. Creates a new link between the main pageflow activity and the first Mode activity in the included flow.
3. Moves all outbound links from the main pageflow to the included flow's Finish activity.
4. Replaces the <modeactivity> nodes from the included flow's document with <activity> nodes.
5. Imports all activities from the included flow into the main flow.
6. Imports all links from the included flow into the main flow.
7. Creates a new process using the dynamically generated pageflow document.

This process is repeated for each included flow.

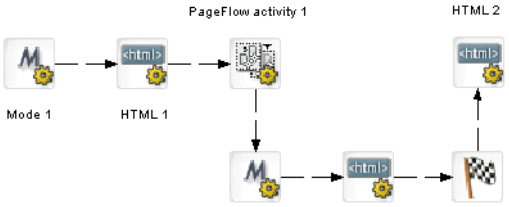
Example

Suppose you have a main pageflow that looks like this:



The Pageflow activity points to this flow:

At runtime, the main pageflow document is manipulated to create a flow that might look like this if it were drawn in the Pageflow Modeler:



JSP activity

Description

The **JSP activity** is a presentation activity that calls a JSP page.

The JSP page called by a JSP activity must handle the creation of its own URL for processing. To get the correct URL for a JSP activity, you need to add these lines to your JSP page:

```
RenderResponse renderResponse =
  (RenderResponse)request.getAttribute( "javax.portlet.response" );
String MYURL = renderResponse.createActionURL().toString();
```

The string `javax.portlet.response` is defined by the Portlet specification. Since exteNd Director provides complete support for the Portlet specification, this string is available to all JSP pages (and servlets) that run within exteNd Director.

Usage

The form you create in your JSP page should look like this:

```
<form name='myform' method='post' action='MYURL'>
```

You can use the JSP Wizard to create JSP page files. If you plan to call a JSP page from a JSP activity, you need to store the JSP page directly within the current WAR file. The JSP activity is not able to access JSP pages stored within a JAR inside the WAR file.

To execute a flow that contains a JSP activity, you need to modify the portlet descriptor generated by the Pageflow Modeler. You must add `charset=ISO-8859-1` to the list of supported mime types:

```
<supports>
  <mime-type>text/html; charset=ISO-8859-1</mime-type>
  <portlet-mode>view</portlet-mode>
</supports>
```

Properties

The properties of the JSP activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
JSP	JSP	Specifies the JSP page to call. The JSP page you specify must be a file in the current WAR. You can use any scoped path to specify the JSP, as long as the path provides the location of the JSP file within the current WAR. For example, you might use a **Session** scoped path to specify a variable containing a path to the JSP file. Most of the time, you will want to use the **Artifact** scope to locate the JSP page. For example, you might specify `Artifact/war://jsps/MyJSP.jsp` as the scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Example

Here is an example of a JSP page that you might call from a JSP activity:

```
<%@ page language="java"
        session="true"
        isThreadSafe="true"
        contentType="text/html; charset=ISO-8859-1" %>

<%
//************************************************************
// User needs to make sure to import "javax.portlet.*;"
%>
<%@ page import="javax.portlet.*" %>

<%
// For the JSP Activity to work, the user must build the correct
// URL for their HTML Form to submit to.

// The HttpRequest object is available to all JSPs by default.
// The user can get the RenderResponse object from the HttpRequest
// at the well-known location of JAVAX.PORTLET.RESPONSE on the
// HttpRequest object.  The user can then build the URL from that
// RenderResponse by calling renderResponse.createActionUrl().
// Whatever the value of that URL is should be placed in the Form.

// Get the RenderResponse
RenderResponse renderResponse = (RenderResponse)request.getAttribute(
"javax.portlet.response" );
String MYURL = "";
try {
    // Build the correct URL for the HTML Form
    MYURL = renderResponse.createActionURL().toString();
}
catch( Exception e ) {
    e.printStackTrace();
}
%>
```

```
<% // Make sure the URL the user built is the "action" attribute
   // of the HTML Form %>
<form method="post" action="<%= MYURL %>">
<% //*******************************************************
%>
    <hr>
    <br><br>
    <p>Enter a value to POST back to Tom: <input name="testValue"/>

    <br>
    <input type="text" name="tomText"/>
    <input type="submit" name="Submit" value="Submit">
    <br><br>
    <hr>
</form>
```

Servlet activity

Description

The **Servlet activity** is a presentation activity that calls a servlet.

At runtime, the Servlet activity calls the doGet() method on the servlet. The doGet() method must return HTML that can be displayed to the user.

The servlet called by a Servlet activity must handle the creation of its own URL for processing. To get the correct URL for a Servlet activity, you need to add these lines to your servlet code:

```
RenderResponse renderResponse =
 (RenderResponse)request.getAttribute( "javax.portlet.response" );
String MYURL = renderResponse.createActionURL().toString();
```

The string `javax.portlet.response` is defined by the Portlet specification. Since exteNd Director provides complete support for the Portlet specification, this string is available to all servlets (and JSP pages) that run within exteNd Director.

Usage

The form you create in your servlet should look this:

```
<form name='myform' method='post' action='MYURL'>
```

You can use the Servlet Wizard to create new servlet Java class files. When you do this, the Servlet Wizard automatically adds the required entries to the web.xml file for the new servlet.

If you plan to call a servlet from a Servlet activity, you need to store the servlet class directly within the current WAR file. The Servlet activity is not able to access servlets stored within a JAR inside the WAR file.

Properties

The properties of the Servlet activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.

Property Inspector tab	Property name	Description
Servlet	Servlet	Specifies the servlet to call. The servlet you specify must map to a servlet name in the web.xml file.
	Content Type	Specifies the content type to display. The content type is specified by means of a scoped path. For example, you might specify `/String/text/xml` as the content type. If you do not specify a content type, the default is `text/html`.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Example

Here is an example of a servlet you might call from a Servlet activity:

```
package com.test;

import javax.portlet.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class myServlet extends HttpServlet {

  static final String CONTENT_TYPE = "text/html";

  // Handle the HTTP GET request
  public void doGet( HttpServletRequest request,
   HttpServletResponse response )
   throws ServletException, IOException {

    response.setContentType( CONTENT_TYPE );
    PrintWriter out = response.getWriter();

//*****************************************************************
    String url = "";
    try {
     // The Servlet Activity must have a certain URL as the value
     // of its HTML Form "action" attribute.
     // Like the HTML Activity, the user will be responsible for
     // building this URL themselves. They must do this, or the
     // flow will not move along to the next activity.

     // The way to get this URL is from the RenderResponse object.
     // The Portlet Spec tells us that the RenderResponse MUST BE      //
AVAILABLE  on the HttpServletRequest
     // object as the JAVAX.PORTLET.RESPONSE attribute.
     // This object will be there for any servlet who wants it.

     RenderResponse renderResponse =
      (RenderResponse)request.getAttribute
       ( "javax.portlet.response" );

    // Once we have the RenderResponse object, the user needs to     // create an
ActionURL from it,
    // and then turn that ActionURL into a String.
```

```
      // This is the URL the user should use as in the HTML form.

      url = renderResponse.createActionURL().toString();
    }
    catch( Exception e ) {
      e.printStackTrace();
    }

    // Now the user needs to put the String as the value of the
    // "action" attribute in the HTML Form.
    // This way the Engine understands that it
    // should move the flow along after a submit if that's what
    // the user wants.

     out.println( "<form name='myForm' method='post' action='" + url + "'>" );
        //*****************************************************************

    out.println( "<p>Servlet MyServlet has received an HTTP
     GET.</p>" );
    out.println( "<p>The servlet generated this page in response
     to the request.</p>" );
    out.println( "<input type='text' name='tomText'/>");
    out.println( "<input type='submit' name='tomSubmit'
     value='blah'/>");
    out.println( "</form>" );
  }


  // Handle the HTTP POST request
  public void doPost( HttpServletRequest request,
    HttpServletResponse response )
    throws ServletException, IOException {

  response.setContentType( CONTENT_TYPE );
  PrintWriter out = response.getWriter();

  out.println( "<p>Servlet has received an HTTP POST.</p>" );
  out.println( "<p>The servlet generated this page in response to
   the request.</p>" );
  out.println( "" );
  }
}
```

Initial Query activity

Description

The **Initial Query activity** obtains the keys of all records from the database that match the user's search criteria. This activity takes an XML document that contains the search terms as input. When the keys have been retrieved from the database, it stores the result set in a record cache.

Properties

The properties of the Initial Query activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Instance Data	Inbound Document Name	Specifies a document that is used as input to this activity. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Get Page activity

Description

The **Get Page activity** retrieves the summary data for a single page of records. This activity uses the set of keys returned by the Initial Query activity to determine which records to include in the summary data.

Properties

The properties of the Get Page activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Instance Data	Inbound Document Name	Specifies a document that is used as input to this activity. The location of the document is specified by means of a scoped path.
	Outbound Document Name	Specifies a document that contains output produced by this activity. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.

Property Inspector tab	Property name	Description
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Get Record activity

Description

The **Get Record activity** retrieves the detail fields for a single record.

Properties

The properties of the Get Record activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Instance Data	Inbound Document Name	Specifies a document that is used as input to this activity. The location of the document is specified by means of a scoped path.
	Outbound Document Name	Specifies a document that contains output produced by this activity. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Record Insert activity

Description

The **Record Insert activity** sends the data for the new record to the database by executing a SQL INSERT statement.

NOTE: If you enable the change log, the SQL INSERT is not actually performed until the Apply Change Logs activity is executed.

Properties

The properties of the Record Insert activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Instance Data	Inbound Document Name	Specifies a document that is used as input to this activity. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Record Update activity

Description

The **Record Update activity** sends the updated data for one record to the database by executing a SQL UPDATE statement. To support optimistic concurrency control, the UPDATE statement includes the original (cached) values in the WHERE clause.

NOTE: If you enable the change log, the SQL UPDATE is not actually performed until the Apply Change Logs activity is executed.

Properties

The properties of the Record Update activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Instance Data	Inbound Document Name	Specifies a document that is used as input to this activity. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.

Property Inspector tab	Property name	Description
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Record Delete activity

Description

The **Record Delete activity** deletes the records selected by the user by executing a SQL DELETE statement. To support optimistic concurrency control, the DELETE statement includes the original (cached) values in the WHERE clause. The Record Delete activity sends a separate DELETE statement for each selected record, but the statements are all executed in a single transaction.

NOTE: If you enable the change log, the SQL DELETE is not actually performed until the Apply Change Logs activity is executed.

Properties

The properties of the Record Delete activity are:

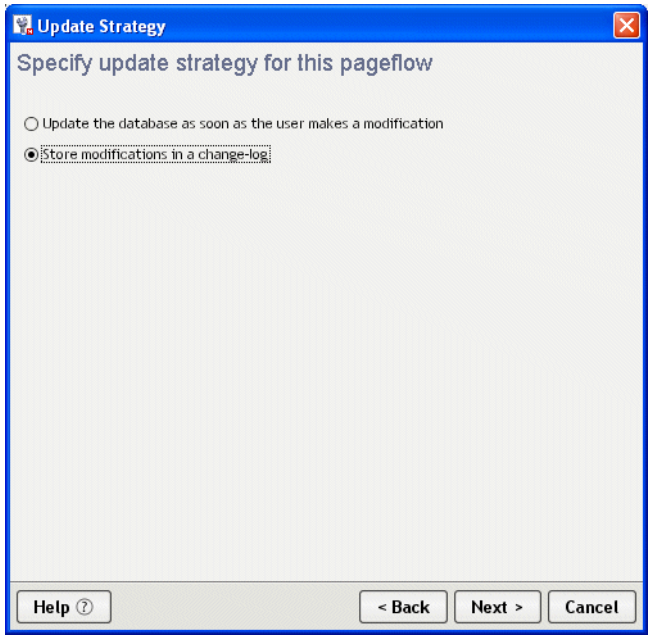
Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Instance Data	Inbound Document Name	Specifies a document that is used as input to this activity. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Apply Change Log activity

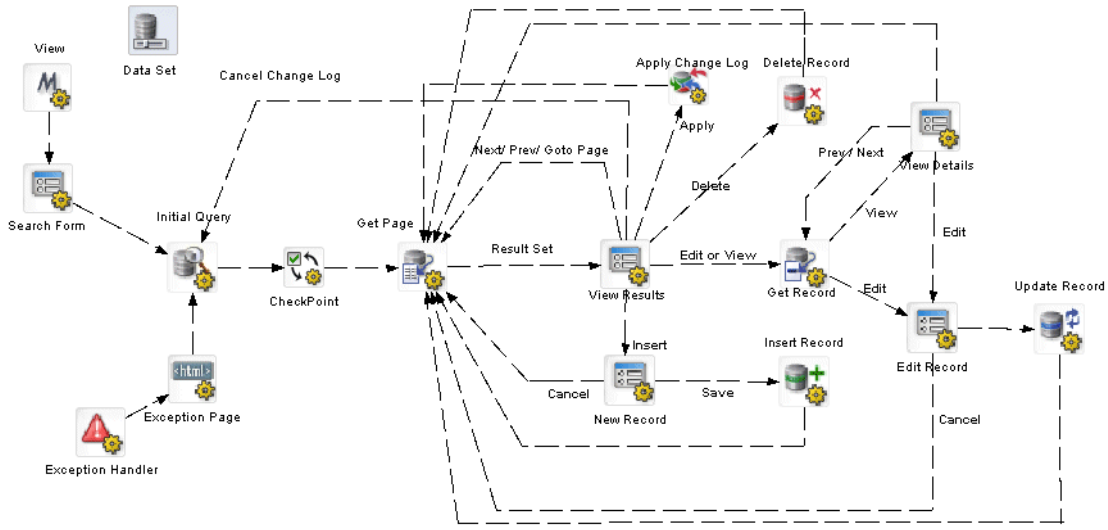
Description

The **Apply Change Log activity** applies all changes from the change log to the database.

When you run the Database Pageflow Wizard, you are given the option to store modifications in a change log, as shown below:



If you indicate that you want to use a change log (as shown above), the wizard includes an Apply Change Log activity in the generated flow:



Properties The properties of the Apply Change Log activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
	Data Set	Specifies the name of a Data Set in the current pageflow. The Data Set you specify must provide all of the settings required to access the database.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Rule activity

Description The **Rule activity** is a system activity that executes a business rule at runtime.

Properties The properties of the Rule activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Rule	Rule ID	Specifies the ID for the rule.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

CheckPoint activity

Description

The **CheckPoint activity** is a directive activity that allows you to specify the processing required to refresh a page within a flow. The CheckPoint activity acts as a transaction marker, indicating the starting point for processing required to rerender the page.

The CheckPoint activity is typically associated with a presentation activity (HTML, Form, or Portlet) within a pageflow. When the user tries to reload the presentation activity, the engine tells the flow to go back to the CheckPoint activity that precedes the presentation activity and continue from that point forward.

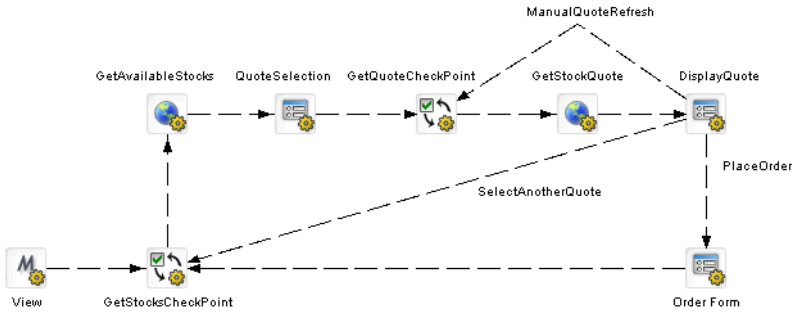
Performance tip To enhance performance, you may not always want to include a CheckPoint activity before a system activity that performs background processing required for a portlet to be rerendered. For example, in a pageflow that invokes a Web Service, you might want to remove the CheckPoint activity so that the Web Service is not executed whenever the user refreshes the page or takes an action on another portlet within the page. If you know that the data associated with a page will not change frequently, then the CheckPoint activity may not be required.

NOTE: An alternative to removing the CheckPoint activity is to use the portlet cache to minimize the number of rerender operations associated with a pageflow.

For more information on using the portlet cache to improve performance in a pageflow, see "Pageflow performance and portlet caching" on page 23.

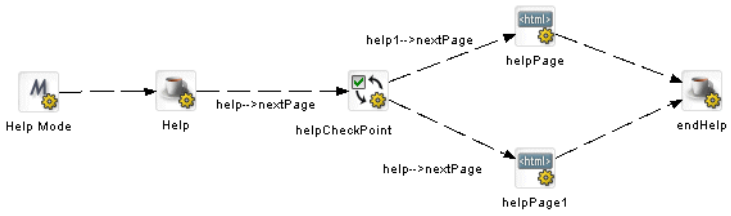
Usage

For example, suppose you have a pageflow that contains a Web Service activity and a Form activity that display the results of the Web Service invocation. The Web Service gets a stock quote based on a ticker symbol. Once the Web Service completes its processing, the Form activity displays the result. Whenever the user reloads the page, the data must be refreshed. To accomplish this, you could create a flow that looks like this:



Note the inclusion of the ManualQuoteRefresh link in this flow. This highlights the difference between a rendering triggered by a request (represented by the ManualQuoteRefresh link) and one triggered by the browser or Portlet container. Also note that the links SelectAnotherQuote and ViewAvailableStocks target the GetStocksCheckPoint. A link directly to the QuoteSelection activity would not cause the GetStocksCheckPoint and GetAvailableStocks activities to run.

In the example shown above, the same page is displayed for each rerender event, but with different data. You might also use a CheckPoint activity in a flow where each rerender event causes a different page to be displayed. For example, in Help mode you might want a flow to display a different help page each time the user requests a reload:



In this example, the links out of the helpCheckPoint activity are evaluated for each refresh to determine which help page should be displayed.

A pageflow may have any number of CheckPoint activities, however, there can be only one CheckPoint activity active at any point in time. At runtime, the CheckPoint activity performs these tasks:

1 Clears any previous CheckPoint activity registered with the process.
2 Registers itself with the process as the CheckPoint activity of record.
3 Sets a flag to indicate that it has just been executed and needs to be kept active.
4 Forwards processing on to any activities that are linked to it.

The logic that deals with how the engine reacts to the inclusion of a CheckPoint activity is built into the presentation activities.

Properties

The properties of the CheckPoint activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Exception activity

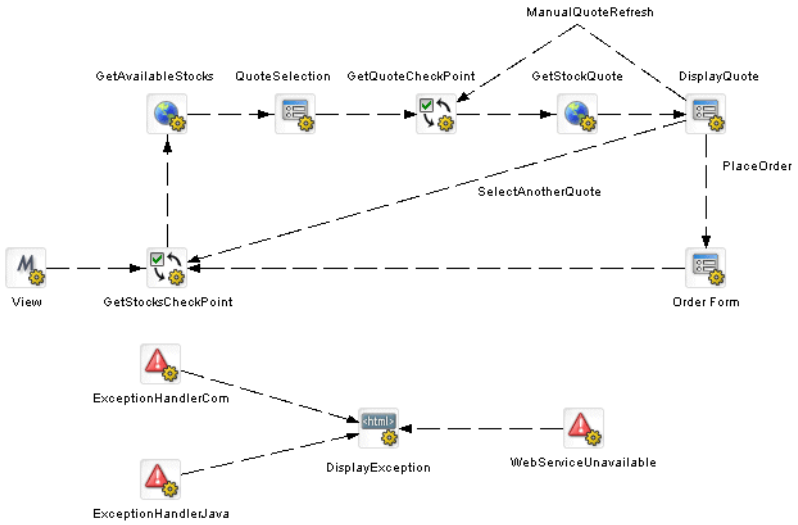
Description

The **Exception activity** is a directive activity that acts as the safety net for a pageflow. Its purpose is to allow you to model how exceptions are handled within the flow. Each Exception activity is the starting point for processing that handles one or more exceptions.

When you create a pageflow, you do not add explicit links that point to the Exception activities. Instead, the links to the Exception activities are generated dynamically at runtime depending on which exception occurred. When an Exception activity gets control at runtime, it simply forwards processing to any activities that are linked to it. Once an Exception Activity has been called, control is not returned to the flow.

Usage

For example, suppose you define a pageflow that contains a Web Service activity. At runtime, it is possible that the Web Service being invoked is not available. To deal with this, you might add an Exception activity to the flow that traps java.rmi.ConnectException. In this case, the engine would forward processing to this activity whenever this exception was encountered. The Exception activity might then forward processing to an HTML activity that displays a meaningful message to the user:



A pageflow may have any number of Exception activities defined for it. A given Exception activity is associated with one or more exception classes. When you place an Exception activity on a flow, you specify its exception string in the **Exception property**. You can specify a complete class name (for example, java.lang,NullPointerException) or a partial name (for example, java.lang) to designate a set of exceptions that this activity should handle.

When an exception is encountered, the pageflow engine tries to match the exception to an Exception activity within the flow. You can instruct the engine to accept exact matches only for a particular Exception activity by setting its **Recursive property** to `false`. However, if you set the Recursive property to `true`, it will look for inexact matches as well, taking into consideration the package hierarchy as well as the inheritance structure.

If a match is found, a dynamic link to the Exception activity is created and a number of scoped paths are created to provide exception information to the custom exception handlers. These are the scoped paths that are created:

Scoped path	Description
Flow/exception/EXCEPTION	Contains a String value that represents the value of the base exceptions.
Flow/exception/STACK	Contains the stack trace for the base exception.
Flow/exception/CLASS	Contains the class name for the base exception.
Flow/exception/MESSAGE	Contains the error message string for the base exception.

If no match is found, or if the exception handler itself throws an exception, an EboProcessException is generated and the exception is handled by the pageflow engine.

Example

Suppose you added an Exception activity that has an exception string of `java.lang` and set the Recursive property to `true`. In this case, suppose the java.lang.NullPointerException were thrown. This exception would be resolved as follows:

1 The engine would compare java.lang.NullPointerException against the list of known exception handlers within the flow and not find a match.

2 The engine would then drop the last segment of the qualified name and try again. This time it would find a match with java.lang and dynamically link to the Exception activity for java.lang.

Suppose the java.security.GeneralSecurityException were thrown. Here's how this exception would be resolved:

1 The engine would compare java.security. GeneralSecurityException against the list of known exception handlers within the flow and not find a match.

2 The engine would then drop the last segment of the qualified name and try again, still not finding a match for java.security.

3 The engine would then drop the last segment of the qualified name and try again, still not finding a match for java.

4 At this point, the engine would try to resolve the exception by looking at the superclass for java.security.GeneralSecurityException, which is java.lang.exception. It would not find a match for java.lang.exception.

5 The engine would drop the last segment of the qualified name and try again. This time, it would find a match with java.lang and dynamically link to the Exception activity for java.lang.

The Exception activity provides you with a flexible architecture for handling exceptions with a flow. If you want a particular Exception activity to catch a wide range of exceptions, you can specify an exception string that has a partial class name and set the Recursive property to true. On the other hand, if you want to exercise more control over what happens when a particular exception is thrown, you can specify a complete class name for the exception string and set the Recursive property to false.

Properties

The properties of the Exception activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Exception	Exception	Specifies a complete class name (for example, java.lang,NullPointerException) or a partial name (for example, java.lang) to designate a set of exceptions that this activity should handle.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Java activity

Description

The **Java activity** is a system activity that executes a Java class within the context of a pageflow. A Java activity allows you to write custom business logic that executes automatically without user intervention.

The Java Activity Wizard helps you create a Java class that can be executed within a pageflow.

For information on using the Java Activity Wizard, see Chapter 10, “Java Activity Wizard”.

Properties

The properties of the Java activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Class Name	Class Name	Specifies the name of the Java class to execute. The package name must also be specified.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

XSL activity

Description

The **XSL activity** is a system activity that uses XSL to transform an XML document.

Each XSL activity includes a property that specifies the XSL to use for the transformation. The activity also specifies an **Input Document** and an **Output Document**. The Input Document specifies the XML to transform. The Output Document specifies the document where the resulting transformation should be placed.

Properties

The properties of the XSL activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
XSL	XSL Document	Specifies a document that contains the XSL to use for the transformation. The location of the document is specified by means of a scoped path.
	Input Document Name	Specifies a document that is used as input to this activity. This document contains the XML you want to transform. The location of the document is specified by means of a scoped path.
	Output Document Name	Specifies a document that contains output produced by this activity. This document contains the result of the XSL transformation. The location of the document is specified by means of a scoped path.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.

Property Inspector tab	Property name	Description
Design UI		Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.

Web Service activity

Description

The **Web Service activity** is a system activity that executes a Web Service.

The Web Service activity only provides support for document-style WSDL files that contain a schema. However, you can create a pageflow that uses an RPC-style Web Service by using a Java activity.

📖 For background information on Web Services, see the chapter on Web Service basics in *Utility Tools*. For details on how to use an RPC-style Web Service in a pageflow, see Chapter 11, “Working with RPC-Style Web Services”.

The quickest and easiest way to create a pageflow that executes a Web Service is to use the Web Service Pageflow Wizard.

📖 For details on using the Web Service Pageflow Wizard, see Chapter 8, “Web Service Pageflow Wizard”.

Properties

The properties of the Web Service activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.

Property Inspector tab	Property name	Description
Document Style	Web Service Input Document Path	Provides instance data for the request being made to the service. The input document is specified by means of a scoped path. Typically, the scope used for the input document is Flow/document.
	Web Service Name	The name of the service, as specified in the WSDL file.
	Web Service Operation	The name of the operation, as specified in the WSDL file.
	Web Service Output Parameter (optional)	The node name of the element returned by the service.
	Web Service Port Type	The port type for the service, as specified in the WSDL file.
	Web Service Return Document Path	Provides instance data for the response returned from the service. The output document is specified by means of a scoped path. Typically, the scope used for the output document is Flow/document.
	Web Service WSDL Document Path	Specifies the name of the WSDL file that describes the Web Service. **TIP:** Fill in this property first when you're working in the Pageflow Modeler. Once the property sheet has a path to the WSDL file, it can automatically fill in many of the other properties associated with the activity.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Composer Service activity

Description

The **Composer Service activity** is a system activity that executes an exteNd Composer service. exteNd Composer *services* typically combine executable units of work called *components* and coordinate the flow of data between them. A typical service might include a series of components that receive an input XML document, perform sophisticated document mappings/transformations, collect information from back-end data sources, execute transactions on mainframes and AS/400s, process error conditions, send context-sensitive e-mail or JMS notifications, and/or return one or more XML response documents to the original requestor(s). By breaking up a service's tasks into discrete components, important benefits—in terms of testing, debugging, code maintenance, and code reuse—can be realized.

The quickest and easiest way to create a pageflow that executes an exteNd Composer service is to use the Composer Pageflow Wizard.

For details on using the Composer Pageflow Wizard, see Chapter 9, “Composer Pageflow Wizard”.

Properties

The properties of the Composer Service activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.
Composer Service	Input Document	Provides instance data for the request being made to the service. The input document is specified by means of a scoped path. Typically, the scope used for the input document is Flow/document.
	Service	Indicates which **exteNd Composer** service to run by specifying a service descriptor in the resource set
	Output Document	Provides instance data for the response returned from the service. The output document is specified by means of a scoped path. Typically, the scope used for the input document is Flow/document.
Copy Scoped Paths	Copy Before	Copies data from one scoped path location to another before the activity is executed.
	Copy After	Copies data from one scoped path location to another after the activity is executed.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Finish activity

Description

The **Finish activity** is a system activity that marks the end of a pageflow. The Finish activity is not required in a pageflow unless it is being used within a another pageflow.

The Finish activity behaves differently depending on how the pageflow is being used:

- In a standalone pageflow, the Finish activity does nothing. The Finish activity is optional within a standalone pageflow.
- When a pageflow is included within another pageflow, the Finish activity simply forwards processing to the next activity within the containing pageflow. In this case, Finish is required so that the pageflow engine knows when to return to the containing pageflow.

Properties

The properties of the Finish activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see “Setting object display properties” on page 81.	

Workflow Return activity

Description

The **Workflow Return activity** is a system activity that tells a pageflow to forward its workitem to a workflow.

To use a pageflow in a workflow you:

- Create a pageflow that lets a user update the persistent workitem that will be passed in from the workflow. Use the Flow scope in the Pageflow Modeler to store the workitem data.
- Add a Workflow Return activity at the point in the flow where you want to return to the workflow and forward to the next workflow activity,
- Add a Pageflow activity in the Workflow Modeler at the point where you want to include the pageflow, and point to the pageflow descriptor. Use the Flow scope in the Workflow Modeler to access the updated workitem.

📖 For more information, see the section on using a pageflow in a workflow in the *Workflow Guide*.

The properties of the Workflow Return activity are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does
	Automatically Close Window	When selected, closes the client window after returning to the workflow.
Design UI	Design UI properties control the design-time appearance of the activity. For more information, see "Setting object display properties" on page 81.	

Usage

If you run a pageflow that contains a Workflow Return activity and there is no workflow queue to forward (or the workflow engine is not running), the pageflow engine displays a message in the browser window:

```
The Workflow Return Activity cannot find a workflow to forward.
```

The user can click the **Continue** button to continue processing.

4 Working with Links

This chapter introduces the links you can use in a pageflow process. It includes these topics:

About links

The following types of explicit links are supported within a pageflow:

- Simple links
- Condition links
- Button links

Pageflows also support **smart linking**. Unlike the other types of links, smart links are not actually drawn on the pageflow diagram.

Links are mutually exclusive The links in a pageflow process are mutually exclusive. At runtime, the pageflow engine will follow only one link out of a particular activity, even if more than one of the link expressions might evaluate to true.

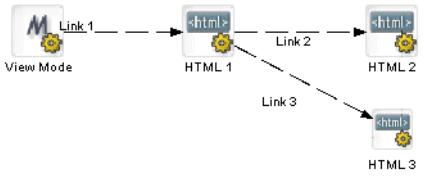
Links are evaluated in order of precedence When a particular activity is linked to more than one activity, the pageflow engine uses **precedence** to determine which link to follow. Each link has a precedence number associated with it. In cases where more than one link expression might evaluate to true, the pageflow engine evaluates the links in precedence order, following the first link that returns true.

What happens when all link expressions return false You can mark a particular link as the **default** path to follow out of a particular activity. When all the link expressions evaluate to false, the pageflow engine follows the default link.

Simple link

A **simple link** represents a simple path from one activity to another. Simple links allow you to specify expressions that evaluate to true or false. When the expression for a simple link evaluates to true, the link is followed to the next activity. When the expression evaluates to false, the path is not followed.

Simple links provide an easy way to choose between multiple target activities. When an activity finishes processing, you can use expressions on simple links to determine which activity should be executed next. For example, you could use expressions on Link 2 and Link 3 in the following pageflow to determine whether HTML 2 or HTML 3 would be displayed after HTML 1:



If you do not specify an expression for a simple link, the pageflow engine automatically moves to the target activity when the source activity finishes. For example, you would not need an expression on Link 2 in this pageflow, since the HTML 1 activity has only one outgoing link:

A simple link has these properties:

Property Inspector tab	Property name	Description
Link	Name	A unique reference to this link that can be accessed in the flow.
	Description	A description of the link intended for the Pageflow Modeler user.
	Copy Scoped Paths	Use to copy scoped data to another scope. The scope gets copied after the link is evaluated and before the target activity is executed. The copy scope operation is performed only when the link expression evaluates to true. 📖 For details on using scoped paths, see the chapter on scoped paths in *Developing exteNd Director Applications*.
	Default	Select to make this path the default. Use if you have multiple links to or from an activity. The default is the path that is used if no other path evaluates to true.
	Expression	Use to build a logical expression that evaluates to true or false. If you do not build an expression, the path will evaluate to true and the path will execute. The expression for a simple link should check the value of a request parameter passed by the source activity. 📖 For details on specifying an expression, see "Creating link expressions" on page 79.
	Precedence	Specify the order in which you want this link to be evaluated. Use if you have multiple links to or from an activity.
Design UI	Design UI properties control the design-time appearance of the link. For more information, see "Setting object display properties" on page 81.	

HTML page with a simple link To initiate a simple link, an HTML page must have a submit button that sets a request parameter to a value specified in the link expression.

For example, if the source activity were an HTML activity, the HTML source for this activity might have a Next button that passes a value of `next` for the parameter named `verb`:

```
<form name="form1" method="post" action="wsrp_rewrite?wsrp-
urlType=blockingAction/wsrp_rewrite">
 <p>Enter your name
    <input type="name" name="name">
    <br><br>
    <input type="submit" name="verb" value="next">
    <input type="submit" name="verb" value="help">
</form>
```

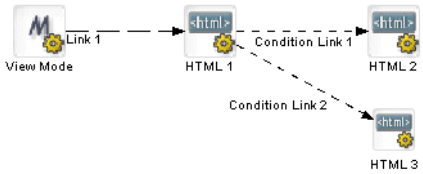
The action for the form specifies the wsrp_rewrite token, an industry-standard token defined by the WSRP specification:

```
action="wsrp_rewrite?wsrp-urlType=blockingAction/wsrp_rewrite"
```

NOTE: This version of exteNd Director does not provide full support for the WSRP protocol. However, support for many WSRP features has been implemented. For example, the wsrp_rewrite token is required in the action for a form.

Condition link

Like a simple link, a **condition link** represents a path from one activity to another. However, a condition link does not have an expression directly associated with it. Instead, it executes a condition macro created by using the Rule Editor. A condition macro executes a set of conditions that contain reusable business logic.



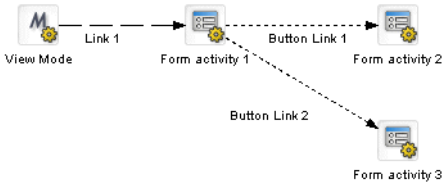
Condition links evaluate to true or false. When a condition link evaluates to true, the link is followed to the next activity. When it evaluates to false, the path is not followed.

A condition link has these properties:

Property Inspector tab	Property name	Description
Condition Link	Name	A unique reference to this link that can be accessed in the flow.
	Description	A description of the link intended for the Pageflow Modeler user.
	Copy Scoped Paths	Use to copy scoped data to another scope. The scope gets copied after the link is evaluated and before the target activity is executed. The copy scope operation is performed only when the link expression evaluates to true. For details on using scoped paths, see the chapter on scoped paths in *Developing exteNd Director Applications*.
	Default	Select to make this path the default. Use if you have multiple links to or from an activity. The default is the path that is used if no other path evaluates to true.
	Condition macro ID	The name of the condition macro XML descriptor, as specified in the Rule Editor.
	Precedence	Specify the order in which you want this link to be evaluated. Use if you have multiple links to or from an activity.
Design UI	Design UI properties control the design-time appearance of the link. For more information, see "Setting object display properties" on page 81.	

Button link

A **button link** creates a dynamic submit button on a form. It is used in conjunction with Form activities and must have a Form activity as its source:



At runtime, a navigation button is placed on the rendered form for each button link that comes out of the Form activity.

The primary benefit of the button link is that it allows you to add navigation to forms dynamically. Whenever you want to change the navigation paths from a form, you can simply add links to the pageflow; you do not need to edit the XHTML form.

Button links specify expressions that evaluate to true or false. When the expression for a button link evaluates to true, the link is followed to the next activity. When the expression evaluates to false, the path is not followed.

Flow Link Region control The button link does not work properly unless the XHTML form associated with the Form activity has a **Flow Link Region** control. The form needs to have only one Flow Link Region control. At runtime, this control is replaced with one or more submit buttons, depending on the number of button links added to the flow. The engine adds a button to the form for each link that does not already have a submit button associated with it.

Key parameter used to determine which button was pressed The pageflow engine uses a special request parameter to determine which button the user pressed on the form. This key is called **novell_link_key** and cannot be changed.

Button link expressions When you add a button link to a pageflow, the Pageflow Modeler automatically defines an expression for a link. The expression checks to see whether the value of the novell_link_key parameter passed on the HttpRequest object is equal to the name of this button link:

```
Request/param/novell_link_key (eq) Button Link 1 [string]
```

If the expression is true, then this link is followed. Ordinarily, you should not change the button link expression or the name of the button link. If you edit the expression or the name, make sure the value used in the expression matches the name for the button link. Otherwise, the link will not function properly at runtime.

Button text At runtime, the engine uses the text specified for the Description property as the text for the dynamically generated submit button.

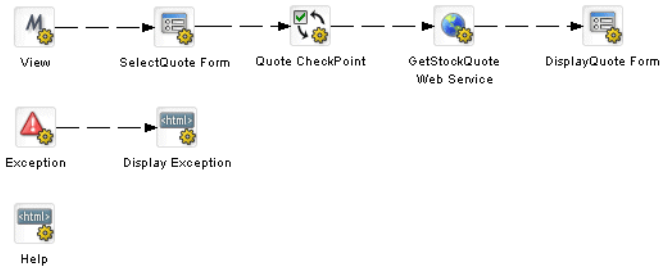
A button link has these properties:

Property Inspector tab	Property name	Description
Button Link	Name	A unique reference to this link that can be accessed in the flow. Ordinarily, you should not change the name of the button link since it is used in the expression for the link. If you edit the name, make sure the value you specify matches the value used in the expression. Otherwise, the link will not function properly at runtime.
	Description	A description of the link that is used as the text for the dynamically generated button.
	Copy Scoped Paths	Use to copy scoped data to another scope. The scope gets copied after the link is evaluated and before the target activity is executed. The copy scope operation is performed only when the link expression evaluates to true. For details on using scoped paths, see the chapter on scoped paths in *Developing exteNd Director Applications*.
	Default	Select to make this path the default. Use if you have multiple links to or from an activity. The default is the path that is used if no other path evaluates to true.
	Expression	Use to build a logical expression that evaluates to true or false. When you add a button link to a flow, the expression is created for you automatically. Ordinarily, you should not change the button link expression. If you edit the expression, make sure the value used in the expression matches the name for the button link. Otherwise, the link will not function properly at runtime. For details on specifying an expression, see "Creating link expressions" on page 79.
	Precedence	Specify the order in which you want this link to be evaluated. Use if you have multiple links to or from an activity.

Property Inspector tab	Property name	Description
Design UI		Design UI properties control the design-time appearance of the link. For more information, see “Setting object display properties” on page 81.

Smart linking

You can use **smart linking** to reduce clutter in a pageflow diagram. Smart links do not actually appear on the pageflow diagram. Instead, they are resolved dynamically at runtime. Smart linking is particularly useful when many activities link to a common activity. For example, you might want to use smart linking to allow all of the activities within a flow to link dynamically to a common Help page:



Smart linking applies to the entire pageflow process, not to individual activities within the process. When smart linking is enabled, all of the activities within the flow can optionally use dynamic linking to access common activities.

NOTE: The pageflow engine also generates dynamic links in the processing of the Exception and CheckPoint activities. Whenever a pageflow includes these types of activities, dynamic linking is used internally, regardless of whether Smart linking is enabled.

Process properties that affect smart linking Smart linking is controlled by the following properties of the process object:

Property Inspector tab	Property name	Description
Process	Use Smartlinking	Indicates whether smart linking is enabled or disabled for the process. When smart linking is enabled, the pageflow engine identifies the target activity for a smart link and spawns a link from the current activity to the target activity. The spawned link may be cached for later use.
	Smartlink Verb	A key that is used to identify the HTTP request parameter whose value will be used to determine the target activity for a dynamic link.

Activities reached through a dynamic link may or may not have outgoing links. In addition, these activities may themselves spawn other dynamic links. In any case, the pageflow engine allows control to return to any of the initiating activities. At runtime, the engine will first try to get a list of explicitly defined links (links specified in the flow diagram) or dynamic links to traverse. If no explicit or dynamic links are available from a particular activity, the engine will try to link back to the last activity executed.

HTML page with smart link To initiate a smart link, an HTML page must have a submit button that sets the Smartlink Verb request parameter to the name of the **activity** that should be the target of the link. Note that this is different from the technique used to initiate a simple link, where the value of the request parameter passed must match a value specified in the link expression.

Here's an example:

```
<form name="form1" method="post" action="wsrp_rewrite?wsrp-
urlType=blockingAction/wsrp_rewrite">
 <p>Enter your name
    <input type="name" name="name">
    <br><br>
    <input type="submit" name="verb" value="next">
    <input type="submit" name="verb" value="help">
</form>
```

The action for the form specifies the wsrp_rewrite token, an industry standard token defined by the WSRP specification:

```
action="wsrp_rewrite?wsrp-urlType=blockingAction/wsrp_rewrite"
```

NOTE: This version of exteNd Director does not provide full support for the WSRP protocol. However, support for many WSRP features has been implemented. For example, the wsrp_rewrite token is required in the action for a form.

XHTML page with smart link An XHTML page might initiate a smart link in the following manner:

```
<xforms:submission
   action="?verb=help"
   id="help"
   method="post"/>
```

When the form is rendered at runtime, the action attribute is marked up automatically, at which point it looks like this:

```
<xforms:submission
   action="wsrp_rewrite?wsrp-urlType=blockingAction&amp;verb=help/wsrp_rewrite"
   id="help"
   method="post"/>
```

You can specify the fully marked-up syntax (as shown above), but this is not necessary, since the complete syntax is generated automatically.

II Tools

Explains how to use the exteNd Director tools to build pageflows

5 Pageflow Modeler

This chapter provides general information about how to create pageflow processes using the Pageflow Modeler. It has these sections:

About the Pageflow Modeler

The Pageflow Modeler allows you to create and save an XML document that can be executed by the pageflow engine at runtime. This document can be called a pageflow process or a process definition. *Pageflow process* refers to the graphical representation of the document; *process definition* more precisely refers to the underlying XML descriptor.

Basic procedure

Creating a pageflow process is a three-step procedure:

1. Use the graphing environment of the Pageflow Modeler to lay out the logic of your process:
   - Add activities
   - Create links between activities
   - Add labels (optional)
2. Use the Property Inspector to configure Activity and Link properties that link the process to the resources of your exteNd Director environment.
3. Save the process in your project resource set. The Pageflow Modeler translates it into the process definition that the pageflow engine can execute.

Creating a view for a pageflow

You can create a custom view of all resource files that are associated with a pageflow. Once you've created the view, you can use display it on the **View** tab.

exteNd Director allows you to export the contents of a view to a JAR. When you export resources from a view, exteNd Director creates a JAR that contains all of the elements in the view, including the directory structure—plus the XML file that defines the view from which the resources were exported. This JAR can then be imported into another resource set.

For details on creating a view for a pageflow, see "Creating a resource view for a pageflow" on page 85.

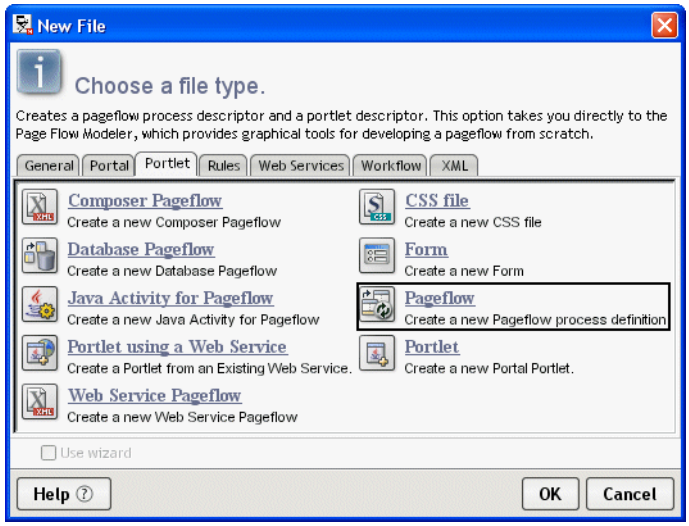
Starting the Modeler

To start the Pageflow Modeler, you create a new process or open an existing one.

➢ **To create a pageflow process:**

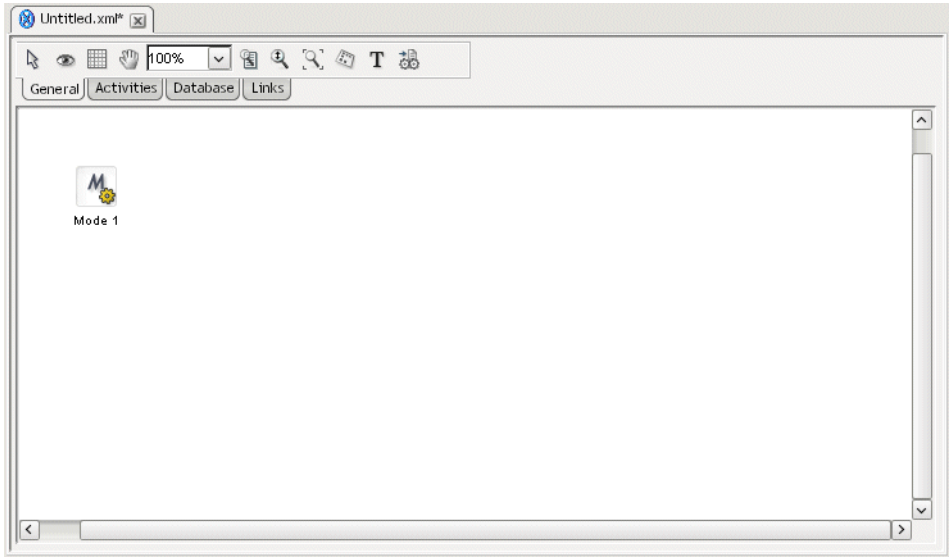
1 With your project open in exteNd Director, select **File>New**.

2 Click the **Portlet** tab and select **Pageflow**:



3 Click **OK**.

The main window of the Pageflow Modeler opens in the editing area. The Mode activity is automatically added to the process:



➢ **To save a pageflow process:**

1 Select **File>Save** from the exteNd Director menu.

If the Pageflow Modeler detects any mistakes in the process, a popup message informs you and asks if you would like to save anyway. This validation occurs on all save events. You can address the warnings later.

2 Click **Yes**.

If this is your first save, the **Save As** dialog appears.

The contents of the **pageflow-process** folder display in your project resource set. Although you can specify another directory, it is recommended that you accept the default.

3 Specify a file name and click **Save**.

➢ **To open a pageflow process:**

1 Select **File>Open** from the exteNd Director menu.

2 Navigate to the **pageflow-process** folder in your project resource set.

3 Double-click to open.

Process properties

You can access the process property sheet to view current information about the process and set properties.

➢ **To access process properties:**

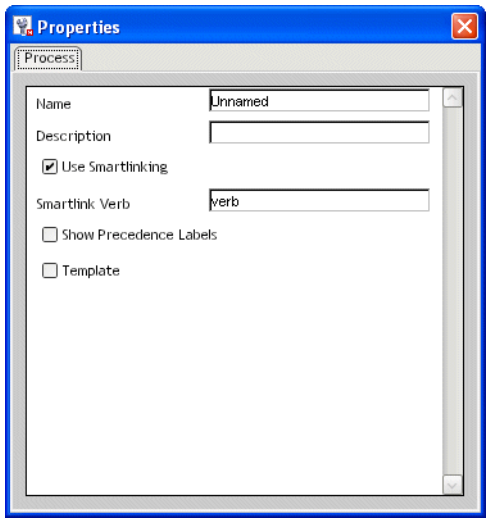
1 Do one of the following:

- Click the Properties icon on the Modeler toolbar:

OR

- With the cursor in the Modeler window, right-click and select **Properties**.

The Property Inspector displays the process properties:



2 Edit or view the process properties:

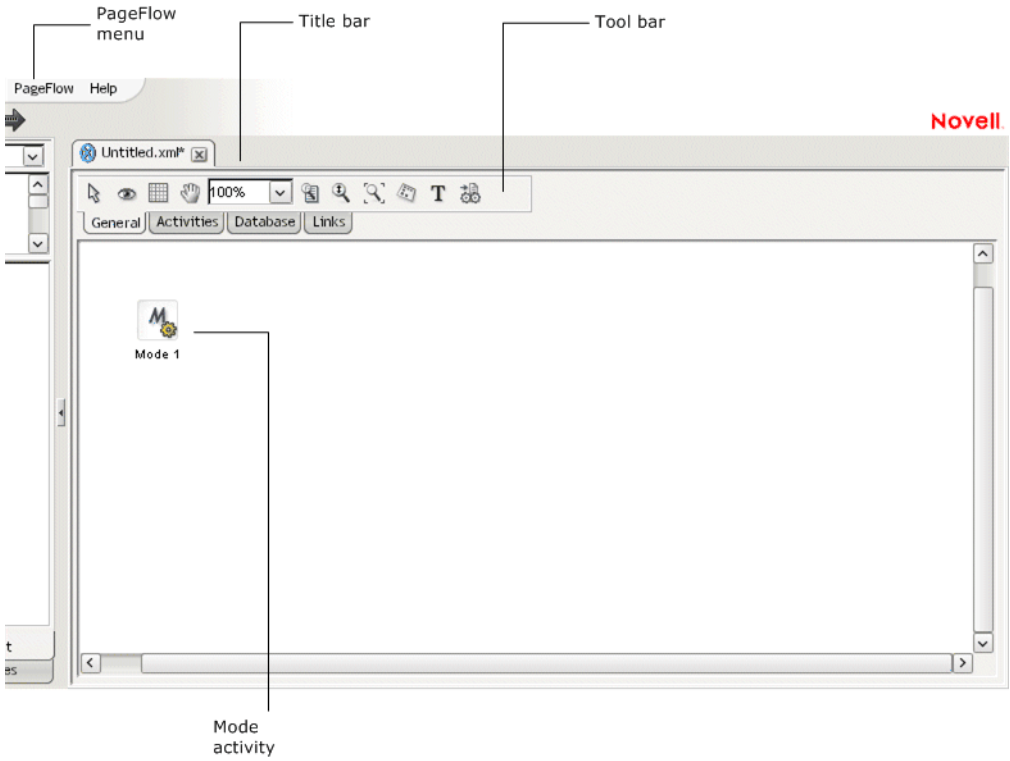
Property	Description
Name	The formal name of the process. The default is the name specified for the process descriptor. If you need to name your processes according to some naming convention, set it here.
Description	A description of the process intended for the Pageflow Modeler user.
Use Smart Linking	Indicates whether smart linking is enabled or disabled for the process. When smart linking is enabled, the pageflow engine identifies the target activity for a smart link and spawns a link from the current activity to the target activity. The spawned link may be cached for later use.
Smartlink Verb	A key that is used to identify the HTTP request parameter whose value will be used to determine the target activity for a dynamic link.
Show Precedence Labels	Specifies whether precedence labels should be displayed in the pageflow graph. When a particular activity is linked to more than one activity, the pageflow engine uses precedence to determine which link to follow. Each link has a precedence number associated with it. In cases where more than one link expression might evaluate to true, the engine evaluates the links in precedence order, following the first link that returns true.
Template	Specifies whether this pageflow can be used as a template for the creation of other pageflows.

About the Modeler window

TIP: To provide more screen space, you can close the Navigation and Output Panes in exteNd Director by clicking the panel view selectors:

Main features

Here are the main features of the Pageflow Modeler:



Navigating, selecting, and moving objects

➢ **To navigate the window:**

1 With the General tab selected, click the **Pan** button on the toolbar to change the cursor to pan mode:

2 Drag the **hand cursor** on the graph area to scroll the graph in the editing area.

You can also use the horizontal and vertical scroll bars to achieve the same effect.

➢ **To select objects:**

1 With the General tab selected, click the **Select** button on the toolbar.

NOTE: Select mode is the default.

2 Click any object to select it.

3 To select multiple objects, click empty space and drag the selection rectangle around the group. You can also select more than one object by holding down the **Ctrl** key while clicking objects with the mouse.

➢ **To move objects:**

1 Select the objects.

2 Drag them to the desired location.

If you move an activity, any links associated with that activity move with it.

Adding activities

An *activity* represents a task in a pageflow. A pageflow activity can present an interface to the user or perform background functions.

Pageflow activity types

Here is a summary of the activities in the Pageflow Modeler. For details about activity properties and usage, click the appropriate item:

Node	Toolbar Icon	Represents
Mode activity		The starting point for a particular mode with a pageflow.
HTML activity		A presentation activity that specifies HTML content to display at runtime.
XML activity		A presentation activity that specifies XML content. The XML activity allows you to use portal styling to control the presentation, which is particularly useful when you want to perform transcoding for wireless applications.
Form activity		A presentation activity that specifies an XHTML file to display at runtime. The XHTML file defines a form for user interaction.
Pageflow activity		Includes a separate pageflow within the current flow.
JSP activity		A presentation activity that calls a JSP page.

Node	Toolbar Icon	Represents
Servlet activity		A presentation activity that calls a servlet.
Initial Query activity		Obtains the keys of all records from the database that match the user's search criteria.
Get Page activity		Retrieves the summary data for a single page of records.
Get Record activity		Retrieves the detail fields for a single record.
Record Insert activity		Sends the data for the new record to the database by executing a SQL INSERT statement.
Record Update activity		Sends the updated data for one record to the database by executing a SQL UPDATE statement.
Record Delete activity		Deletes the records selected by the user by executing a SQL DELETE statement.
Apply Change Log activity		Applies all changes from the record cache to the database.
Rule activity		A system activity that executes a business rule at runtime.
CheckPoint activity		A directive activity that allows you to specify the processing required to refresh a page within a flow. The CheckPoint activity acts as a transaction marker, indicating the starting point for processing required to rerender the page.
Exception activity		A directive activity that acts as the safety net for a pageflow. Each Exception activity is the starting point for processing that handles one or more exceptions.
Java activity		A system activity that executes a Java class within the context of a pageflow.
XSL activity		A system activity that uses XSL to transform an XML document.

Node	Toolbar Icon	Represents
Web Service activity		A system activity that executes a Web Service.
Composer Service activity		A system activity that executes a exteNd Composer service.
Finish activity		A system activity that marks the end of a pageflow. The Finish activity is not required in a pageflow unless it is being used within another pageflow.
Workflow Return activity		A system activity that tells a pageflow to forward its workitem to a workflow.

Using activities

➢ **To add an activity:**

1 Click the **Activities** tab.

The activity toolbar displays:

2 Select an activity type.

See "Pageflow activity types" on page 70 for more information about the activity types.

3 Click anywhere on the graph to place the object.

➢ **To add a database activity:**

1 Click the **Database** tab.

The activity tool bar displays:

2 Select an activity type.

See "Pageflow activity types" on page 70 for more information about the activity types.

3 Click anywhere on the graph to place the object.

➢ **To delete an activity:**

1 Select the activity.

2 Press the **Delete** key or right-click and select **Delete**.

➢ **To access activity properties:**

1 Select the activity.

2 Right-click and select **Properties**, or click the Eye icon on the **General** tab.

3 Click the appropriate tab in the property sheet.

📖 The properties are specific for the activity type. For more information, see "Pageflow activity types" on page 70.

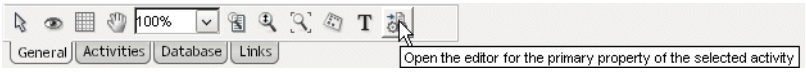
➢ **To open the source file for an activity:**

1 Select the activity.

2 Click the **General** tab.

The General toolbar displays:

3 Click the **Open Primary Editor** icon on the toolbar.



The source file is displayed in an editor.

Adding links

A *link* is a single logical path from one activity to another activity. Links are represented by arrows in the Pageflow Modeler.

Pageflow link types

Below is a summary of the activities in the Pageflow Modeler. For details about usage and setting link properties, click on the appropriate item:

Link type	Toolbar icon	Represents
Simple link		A logical path from one activity to one or more other activities. On the link properties you specify the routing logic using the link expression builder.
Condition link		A logical path from one activity to one or more other activities. On the link properties you specify a Condition Macro that you created using the Rule Editor.
Button link		Creates a dynamic submit button on a form. It is used in conjunction with Form activities and must have a Form activity as its source. On the link properties you specify a name and a description for link. When you add a button link to a flow, the expression is created for you automatically using the name you assigned to the link.

➢ **To create a link:**

1 Click the **Links** tab.

The link toolbar displays:

2 Select a link type.

3 Click an activity and drag from the starting activity to the target activity, then release the mouse button.

You can use link segments to enhance the legibility and appearance of the pageflow. For more information on link segments, see "Drawing a link segment" on page 74.

➢ **To delete a link:**

1 Select the link.

2 Press the **Delete** key or right-click and select **Delete**.

➢ **To access link properties:**

1 Select the link.

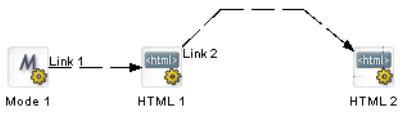
2 Click the Eye icon on the **General** tab or right-click and select **Properties**.

3 Click the appropriate tab in the property sheet.

Properties are specific for the link type. For more information, see "Pageflow link types" on page 73.

Drawing a link segment

A *segment* is the line between two points of a link. Segments are purely cosmetic. You can use them to enhance the legibility and appearance of the pageflow. Here is a pageflow diagram that shows a link (Link 2) with three segments:



➢ **To draw a segment:**

1 Click any point on a link and drag in any direction.

The point you drag becomes the bend in the link.

2 Release the mouse button at the point you want to end the segment.

3 Click again and drag to add another segment.

4 For the final segment, drag to the destination activity and click it.

Using scoped paths

Scoped paths allow you to associate data with pageflow activities and links. exteNd Director provides a set of predefined scoped paths that you can access using the scoped path dialogs and the Scoped Path Navigator.

For background information about scoped paths, see the chapter on working with scoped paths in *Developing exteNd Director Applications.*

This section describes how to:

- Associate scoped paths directly with the primary property for an activity
- Copy scoped paths before or after an activity or a link executes in the flow
- View scoped paths used in a flow
- Copy scoped paths to the clipboard

For information about scoped path usage with examples, see "Using scoped paths in a pageflow" on page 21.

Associating a scoped path with an activity

You can use a scoped path to associate an object with an activity. To do this, you specify a scoped path as the value for the activity's primary property. You can do this with any of the following types of activities:

- Form activity
- HTML activity
- XML activity
- Pageflow activity
- JSP activity
- Servlet activity
- Web Service activity
- Composer Service activity
- XSL activity
- Rule activity
- Java activity

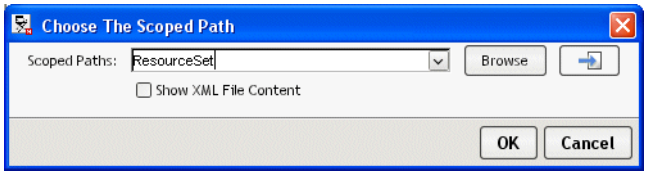
➢ **To associate a scoped path with an activity's primary property:**

1 Select the activity, then right-click and select **Properties**.

2 In the Property Inspector, click the tab that has the primary property. This tab typically has the same name as the activity. For example: the HTML activity has an HTML tab where you can set the primary property, and the Form activity has a Form tab.

3 Click the primary property for the selected activity. For details, see "Pageflow activity types" on page 70.

The Choose The Scope dialog displays:



The dialog displays the default predefined path for this activity type.

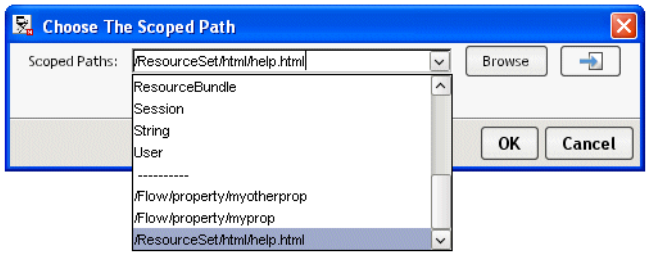
4 Use one of these methods to specify the path:

- Type the path in the text box.

  This requires you to know the syntax for the scoped path. For more information, see the section on predefined scoped paths in *Developing exteNd Director Applications.*

  OR

- Select any one of the paths already defined in this flow; these paths appear at the bottom of the dropdown list below the dashes.

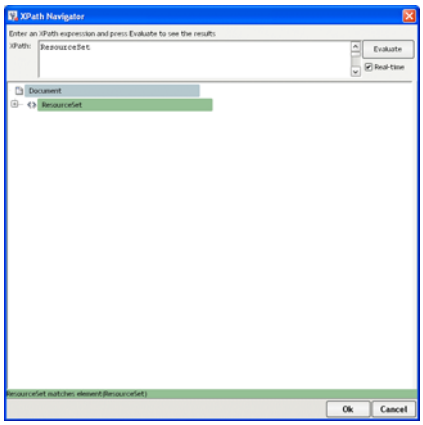


TIP: If you choose a current path that accesses the resource set, you can display the contents of the file by clicking the icon next to the **Browse** button.

OR

- Select one of the predefined scopes at the top of the dropdown and click **Browse**.

  When you click **Browse**, the Scoped Path Navigator displays.



  The Scoped Path Navigator is an interactive tool that allows you to specify the path using a tree view (where applicable) and the XPath expression builder. The available options depend on the scoped path you selected.

  📖 For more information on using the Scoped Path Navigator, see the section on using the scoped paths in *Developing exteNd Director Applications*.

5 After you specify the path, click **OK**.

Editing a scope To edit an existing scoped path, repeat the procedure.

Copying scoped paths

With the exception of the Mode and Finish activities, you can copy scoped paths before or after the execution of any activity, and copy scoped paths after the execution of a any link.

📖 For design considerations that relate to copying scoped paths, see the section on copying scoped paths in *Developing exteNd Director Applications*.

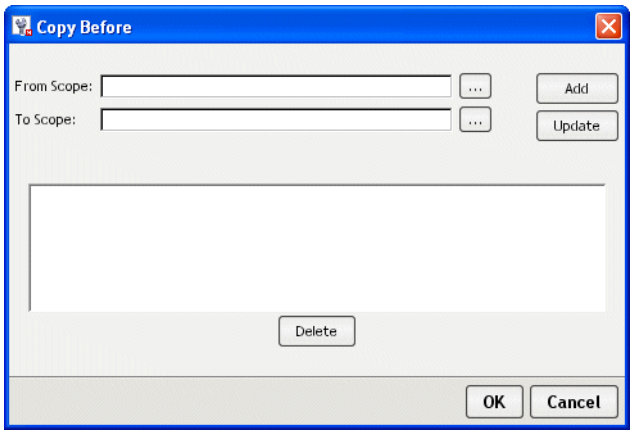
➢ **To copy a scoped path before or after an activity:**

1 Select the activity, then right-click and select **Properties**.

2 In the Property Inspector, click the **Copy Scoped Paths** tab:

- To copy the scoped path **before** execution, click the link **Edit Scoped Paths** under **Copy Before.**
- To copy the scoped path **after** execution, click the link **Edit Scoped Paths** under **Copy After**.

The appropriate Copy dialog (Copy Before or Copy After) displays:



3 Click the ellipsis for **From scope**.

The dialog displays the default predefined scope for this activity type. If you want to use another scope, select it from the dropdown list.

4 To specify the path, use one of the methods described in Step 3 in the preceding procedure ("To associate a scoped path with an activity's primary property:" on page 75).

5 Click the ellipsis for **To scope**. Repeat the procedure for specifying the From scope.

6 Click **Add** and then **OK** to exit the Copy Scope dialog.

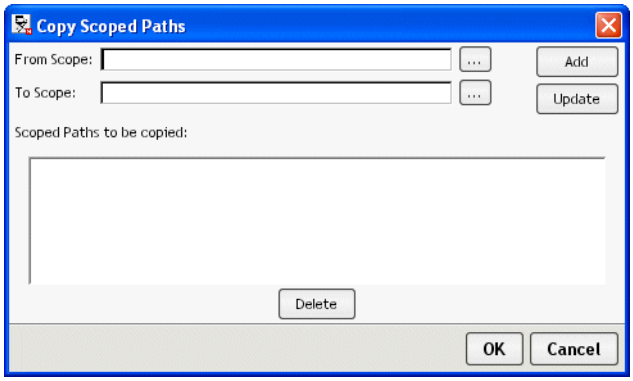
To edit a scope To edit an existing scope, repeat the procedure—but click **Update** instead of Add.

➢ **To copy a scoped path after the execution of a link:**

1 Select the link, then right-click and select **Properties**.

2 In the Property Inspector, click the link **Edit Scoped Paths** under **Copy Scoped Paths.**

The Copy Scoped Paths dialog displays:



3 Click the ellipsis for **From scope**

The dialog displays the default predefined scope for this activity type. If you want to use another scope, select it from the dropdown list.

4 To specify the path, use one of the methods described in Step 3 in the preceding procedure ("To associate a scoped path with an activity's primary property:" on page 75).

5 Click the ellipsis for **To scope**. Repeat the procedure for specifying the From scope.

6 Click **Add** and **OK** to exit the Copy Scope dialog.

7 On the Copy Scope dialog click **Add** and **OK** to exit the Copy Scope dialog.

To edit a scope To edit an existing scope, repeat the procedure—but click **Update** instead of Add.

Accessing scoped paths

You can view all of the current paths used for a selected object in the flow or for all objects in the flow.

➢ **To access current scoped paths for an activity or link:**

1 Select the object.

2 Right-click and choose **Scoped paths**.

The Available Scoped Paths dialog displays.

➢ **To access all current paths in the flow:**

1 From the **Pageflow** menu

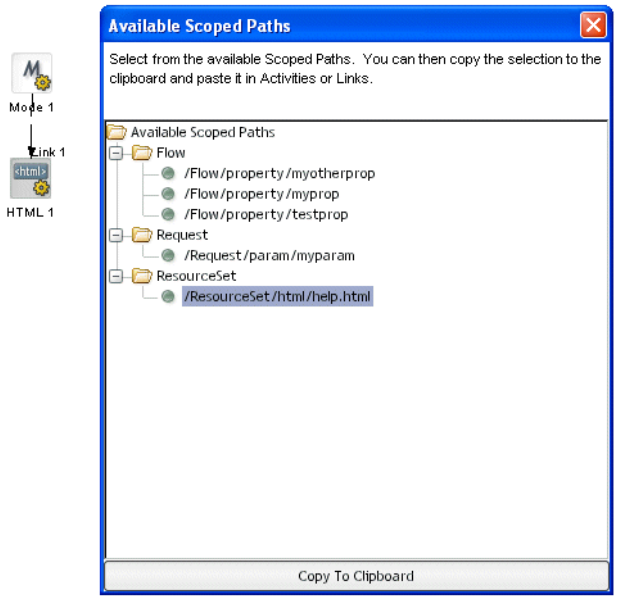
- Select **View Scoped Paths**.

  OR

- With no objects selected, right-click and choose **Scoped paths**.

The Available Scoped Paths dialog displays.

2 Click a scoped path. Notice that the objects that use it are highlighted:



Copying a scoped path to the clipboard

From the Available Scoped Paths dialog you can copy a selected path to the clipboard and paste it wherever you want.

➢ **To copy a scoped path to the clipboard:**

1 Access the scoped paths using one of the methods described in the preceding section.

2 Select the path you want to copy.

3 Double-click the path, or click **Copy to Clipboard** at the bottom of the dialog.

4 Go to where you want to copy the path and press **Ctrl+V**.

Creating link expressions

Link expressions let you use runtime values to choose between multiple target activities. When an activity finishes processing, you can use expressions to determine which activity should be executed next. The expression builder allows you to test runtime values in simple or compound expressions that evaluate to true or false. Expressions apply to simple links and button links only.

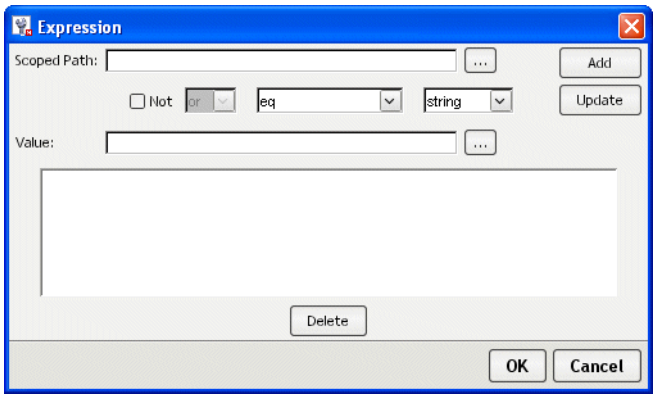
For more information on simple links, see "Simple link" on page 55. For more information on button links, see "Button link" on page 58.

➢ **To specify a link expression:**

1 Select a simple link or a button link, right-click, and select **Properties**.

2 In the Property Inspector, select **Edit Expression**.

The Expression dialog displays:



3 Click **...** to access a scope.

4 To specify the path, use one of the methods described in Step 4 in the procedure "To associate a scoped path with an activity's primary property:" on page 75.

5 In the **Value** field, specify the value to test against the scoped data.

6 Select the operator from the data operator dropdown list.

7 Select the data type (which must match the scoped data) in the data type dropdown.

- For **Boolean**, you must enter **true** or **false**.
- For **Timestamp**, use this format:

```
MM/DD/YYY < optional: HH:MM:SS>
```

The expression builder validates your entry.

8 Click **Add**.

This adds the expression to the text area at the bottom.

9 To add additional clauses to the expression, select the appropriate item from the logical operator dropdown, complete the clause, and click **Add.** Add as many clauses as you need.

NOTE: To negate the current expression, enable the **Not** check box.

10 To edit or delete an expression, click **Update** or **Delete**.

11 Click **OK** when you're done.

Validating a process

You can validate a process at design time whenever you choose. The modeler analyzes the process structure and displays any errors encountered. Note that the validation applies to the design-time process structure only.

➢ **To validate a process:**

- Select **Validate Process** from the **Pageflow** menu.
  OR
- With or without any objects selected, right-click and select **Validate Process**.

Adding and manipulating text labels

Labels are separate objects in the graphing environment and have their own property sheets. Labels have two forms:

- Floating labels
- Attached labels

Floating labels

Floating labels are simply text you place on your pageflow graph and have no association with another pageflow graph object. Titles, version numbers, notes, and legends are all good uses for floating labels.

➢ **To create a floating label:**

1. From the Pageflow Modeler toolbar, select the **Text Block** button:

2. Click the location on the graph where you want the label to appear.
   The label appears as a box with the text **Untitled** inside.
3. From the toolbar, select the arrow button.
4. Double-click inside the new label and edit the text.
5. Click outside the label to save your changes.

Attached labels

Most labels are associated with an activity or link and are called attached labels. By default, activities and links start with a label.

➢ **To edit an attached label:**

1. Right-click an activity icon or link.
2. From the popup menu, select **Create Label**.
   A label appears directly below the activity or link.
3. Double-click inside the label and edit the text.
4. Click outside the label to save your changes.

An activity or link can have many labels. You can reposition the attached label by dragging it to a new location. Note the line that appears as you drag. This line indicates the activity or link the label is attached to.

➢ **To format any label:**

1 Right-click the label and select **Properties**.

The Property Inspector displays showing the current formatting properties.

2 Make your changes. Properties are described in "Setting object display properties" on page 81.

Changes are saved as you make them.

➢ **To delete a label:**

1 Select the label.

2 Press the **Delete** key.

Setting object display properties

Each activity, link, and label has a set of properties associated with it. Select and right-click an element, and the properties of the element are displayed in the Property Inspector. The table below describes the graphical properties found on some or all of the elements:

Graphical properties	Description
Arrowhead Height	Customize the height (thickness) of the arrowhead for the selected links.
Arrowhead Width	Customize the width (length) of the arrowhead for the selected links.
Border Color	The color of the square outlining the activity. Click the color bar to display a standard color selection dialog.
Color	The background color of the activity. Click the color bar to display a standard color selection dialog.
Font	Click the data area to bring up a standard text formatting dialog.
Height	(Read-only) Height of the activity in pixels. You can enlarge or shrink the activity by dragging its handles.
Margin Height	For labels, the amount of space on the top and bottom between the text and the bounding box.
Margin Width	For labels, the amount of space on the left and right sides between the text and the bounding box.
Show Border	When enabled, displays a square outline around the activity, even when the background color is set to transparent.
Style	Choose a solid line or one of several dashed patterns from the dropdown list. Your choice affects only the currently selected link destination. To change several destinations at once, hold down the Shift key and then click each one. Select a style from the Property Inspector.
Text color	The font color of the label. Click the color bar to display a standard color selection dialog.
Transparent	Overrides the color setting and makes the activity background transparent.
Width	(Read-only) Width of the activity in pixels. You can enlarge or shrink the activity by dragging its handles.

Graphical properties	Description
X Center	When the pageflow process is first created, the origin (0, 0) is the bottom-left corner of the graph. The graph automatically resizes in all directions as you create and drag items around. When this happens, the origin does not reset itself to the new bottom-left corner; it remains fixed. A positive value is the number of pixels above the origin the vertical center of the icon is currently located. A negative value indicates a position below the origin. Enter a new value to have the Pageflow Modeler automatically move the activity to the vertical position specified.
Y Center	When the pageflow process is first created, the origin (0, 0) is the bottom-left corner of the graph. The graph automatically resizes in all directions as you create and drag items around. When this happens, the origin does not reset itself to the new bottom-left corner; it remains fixed. A positive value is the number of pixels to the right of the origin the horizontal center of the icon is currently located. A negative value indicates a position to the left of the origin. Enter a new value to have the Pageflow Modeler automatically move the activity to the horizontal position specified.

Using the layout features

The *layout* is the arrangement of the activities, links, and labels in your graph. The Pageflow Modeler has a sophisticated layout feature that can completely rearrange your graph to maximize readability and minimize space.

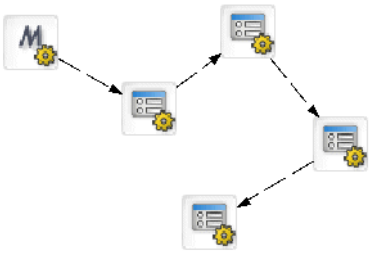
You can specify whether you want the new arrangement to have a horizontal or vertical orientation and whether you want link links drawn as diagonal lines or composed of perpendicular segments.

There are two kinds of layout: *full layout* and *incremental layout.*

Full layout

Full layout gives the Pageflow Modeler great freedom to move activities, links, and labels around the graph.

For example, this figure displays a hand-arranged layout:



This figure shows the result of applying a full layout to the hand-arranged layout:

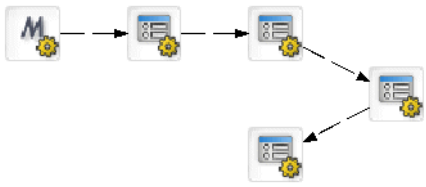
➢ **To apply a full layout:**

- Choose **Pageflow>Full Layout** from the exteNd Director menu:
  OR
- Press **Ctrl+Shift+L**.

Incremental layout

Incremental layout tries to make a graph more attractive and organized but also tries to keep the basic design of your hand-arranged layout.

This figure shows the result of applying an incremental layout to the hand-arranged layout shown above:



TIP: Try a layout on your graph. Select **Edit>Undo** to back out of a layout you don't want.

➢ **To apply an incremental layout:**

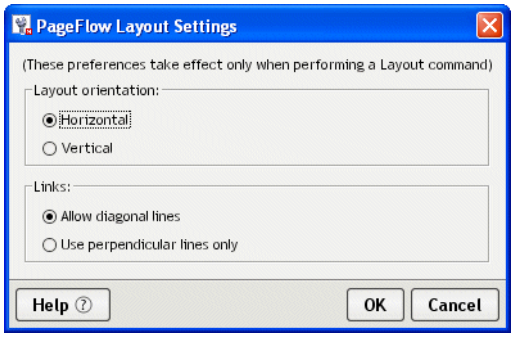
- Choose **Pageflow>Incremental Layout** from the development environment menu.
  OR
- Press **Ctrl+Shift+M**.

Setting preferences

You can set layout preferences for a pageflow graph. These preferences let you specify whether you want the new arrangement to have a horizontal or vertical orientation and whether you want link links drawn as diagonal lines or composed of perpendicular segments.

➢ **To set layout preferences:**

1 Choose **Pageflow>Layout Settings** from the development environment menu.



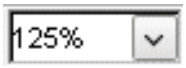
2 Select an orientation.

3 Select a link style.

4 Click **OK**.

Using the zoom features

The Pageflow Modeler gives you four ways to zoom:

Zoom type	Toolbar icon	Description
Standard zoom		Allows you to pick from a list of common zoom percents
Marquee zoom		Allows you to drag and select a portion of the graph area to be zoomed to fill the graph window
Interactive zoom		Allows you to zoom up or down by dragging up or down on the graph
Fit in window		Allows you to zoom the graph window to show all the activities in the graph. To do this, it shrinks or enlarges the content of the current document to fit in the graph window.

Using the grid features

The Pageflow Modeler includes a drawing grid that works much like the grid in any graphics program:

- When the grid is **visible**, the corners of activity and label objects stick to the intersections of horizontal and vertical grid lines. This makes it much easier to line up objects and use consisting spacing.
- When the grid is **invisible**, objects can be positioned without constraint in one-pixel increments.
- Turning the grid on does not reposition existing items to align with the grid.

➢ **To turn the grid on or off:**

- Choose **Pageflow>Toggle Grid** from the exteNd Director menu.

  OR

- Click the **Grid** button from the Pageflow Modeler toolbar:

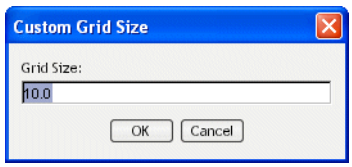
  OR

- Press **Ctrl+Shift+G**.

➢ **To change the spacing of grid lines (grid size):**

1. Choose **Pageflow>Grid Size** from the exteNd Director menu.

   The Grid Size dialog displays:

2. Enter a value in pixels from 7.5 to 1000.

   The default is 10.

3. Click **OK**.

Using the Bird's Eye View

The Bird's Eye View is a popup window that gives you a view of the entire pageflow graph to help you find your way around in a large graph. The Bird's Eye View window:

- Appears when you click the Bird's Eye View button on the Pageflow Modeler toolbar:

- Stays on top as you work in the graph area
- Is dismissed by clicking the X button in the upper-right corner
- Can be resized by dragging the corners
- Indicates the area visible in the graph window with a blue outline box

You can use the blue outline box of the Bird's Eye View window to do several useful things:

Doing this	Has this effect
Clicking outside the outline box	Centers the outline box on the point clicked and pans the graph area to correspond to the new location of the outline box
Dragging outside the outline box	Draws the outline box in the new location and pans the graph area to correspond to the new location of the outline box
Dragging inside the outline box	Pans the outline box and pans the graph area to correspond to the new location of the outline box
Dragging a corner of the outline box	Resizes the outline box and zooms the graph area to correspond to the new size of the outline box

Creating a resource view for a pageflow

You can create a custom view of all resource files that are associated with a pageflow. Once you've created the view, you can display it on the **View** tab.

exteNd Director allows you to export the contents of a view to a JAR. When you export resources from a view, exteNd Director creates a JAR that contains all of the elements in the view, including the directory structure—plus the XML file that defines the view from which the resources were exported. This JAR can then be imported into another resource set.

TIP: To get to the **View** tab, you need to first click the **Resources** tab in the exteNd Director Navigation Pane.

📖 For more information on views, see the chapter on working with views in *Developing exteNd Director Applications*.

➢ **To create a resource view for a pageflow process:**

- Choose **Pageflow>Create Resource View** from the exteNd Director menu.

  The view created has the same name as the pageflow process descriptor. The view definition is stored as an XML document in the **my-views** folder within the resource set.

Deleting a pageflow

To remove a pageflow, you need to delete the pageflow process descriptor file. When you delete the descriptor file, any files referenced by the pageflow (such as Java source files, HTML files, XML files, and other resources) are not removed. Therefore, if you want to remove these files, you need to delete them by hand.

When you create a pageflow, the Pageflow Modeler saves a *portlet fragment deployment descriptor* in the resource set. When you delete a pageflow, you need to delete this descriptor and unregister the portlet as well.

📖 For more information about deleting portlets, see the section on deleting portlets in the *Portal Guide*.

6 Form Designer

This chapter introduces the Novell exteNd Director Form Designer and describes how to use it to create and modify XForms 1.0-compliant Web forms. It includes these sections:

About XForms

XForms provide a robust, standards-based way to define Web forms. The advantages of the XForms standard include:

- Separate data, logic, and presentation modules
- A powerful event model (so that you don't have to use a lot of scripting for client-side validation or calculations)
- A way to process XML data

XForms cannot run as standalone applications. They are designed to run as components within a host language like XHTML. In Novell's implementation, they run within the context of a pageflow application.

About the Form Designer

The Form Designer provides a graphical environment for developing XForms 1.0-compliant Web forms.

The Form Designer is divided into these tabs:

Tab	Description
Form	Lets you define the form's user interface. You can graphically: ◆ Lay out and style form controls ◆ Bind form controls to data ◆ Define events and actions for form controls 📖 For more information, see "Defining the presentation" on page 94.

Tab	Description
Model	Lets you define the form's model elements. You can: ◆ Create and edit models ◆ Create and edit instance data ◆ Set up data constraints 📖 For more information, see "Working with model elements" on page 109.
Source	Launches a powerful XML source editor. 📖 For more information on working using the XML Editor, see the chapter on the XML Editor in *Utility Tools*.
XForms Preview	Lets you run a form in test mode. 📖 For more information, see "Testing forms" on page 130.

Starting and stopping the Form Designer

To start the Form Designer, you:

- Open an existing form (see To open an existing form: next)
  OR
- Create a new form (see "Creating forms" on page 88)

➢ **To open an existing form:**

1 Select **File>Open**.
2 Navigate to the location of the existing form (usually in the project's \data\form directory).
3 Click the form file and choose **Open**. (Optionally, you can double-click the form.)

Creating forms

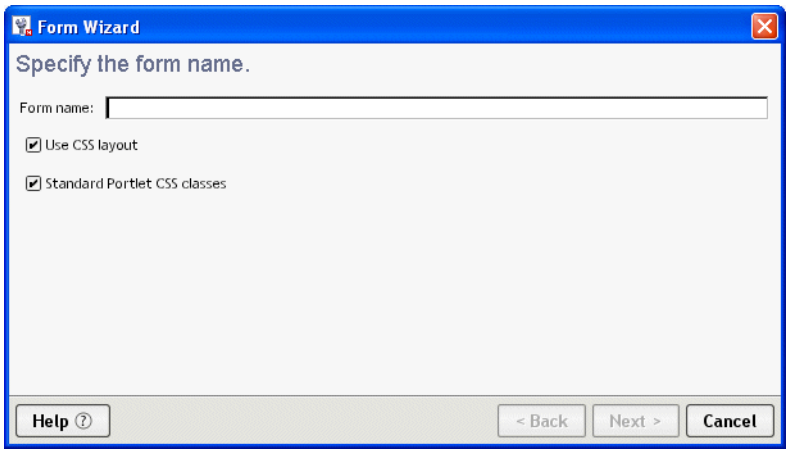
You use the Form Wizard to create either an initial unbound form or a data-bound form based on:

- Schema
- Sample XML data (instance data)

Other strategies To create a complete pageflow including input and output forms from any other source, you'll launch a different wizard. For more information on launching other wizards that generate XForms within the scope of a pageflow application, see Chapter 7, "Database Pageflow Wizard", Chapter 8, "Web Service Pageflow Wizard", and Chapter 9, "Composer Pageflow Wizard".

➢ **To create a form using the Form Wizard:**

1 Open an exteNd Director project.
TIP: You can't start the Form Wizard unless you have a project that contains a resource set open.
2 Select **File>New**.
3 On the **Portlet** tab, choose **Form** and click **OK**. (Alternatively, you can double-click **Form**.)
The first panel of the Form Wizard displays:

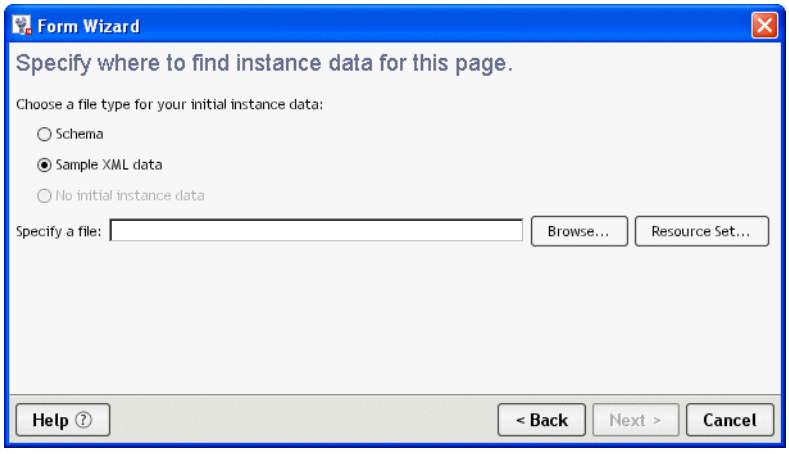


4 Specify the following options:

Option	What to do
Form name	Specify a name for the form. You don't need to specify any extension.
Use CSS layout	Check this box when you want the wizard to use absolute positioning to layout the controls on the form, otherwise, uncheck this box. If you choose not to use CSS layout, absolute positioning is not available. For more information about how the wizard uses CSS layout, see "About the wizard-generated forms" on page 91.
Standard portlet CSS classes	Check this box when you want the wizard to use the WSRP CSS classes for fonts and colors. (The defaults are defined in the wsrp-classifications.xml file located in the Common\Resources\CSSClassifications directory). When checked, these classes are used in combination with the portal theme files at runtime for styling the page. Uncheck this box if you want to define your own colors and fonts.

5 Click **Next** to proceed.

The following wizard panel displays:



6 Specify one of the following file types for your initial instance data:

Option	What to do
Schema	Choose this option if you want the wizard to generate controls that are bound to the instance data nodes created from the elements of an existing schema. The schema type of each instance data node is automatically specified.
Sample XML data	Choose this option if you want the wizard to generate controls that are bound to instance data nodes. If you want to specify formatting for the displayed data values, you will have to add schema type information manually.
No initial instance data	Choose this option to create XForms with no data-bound controls.
Specify a file	Specify the name and location of the file containing the schema or the sample XML data: ◆ Browse—Choose this option when the file is located on disk. ◆ Resource Set—Choose this option when the file is located in a resource set within the currently open project.

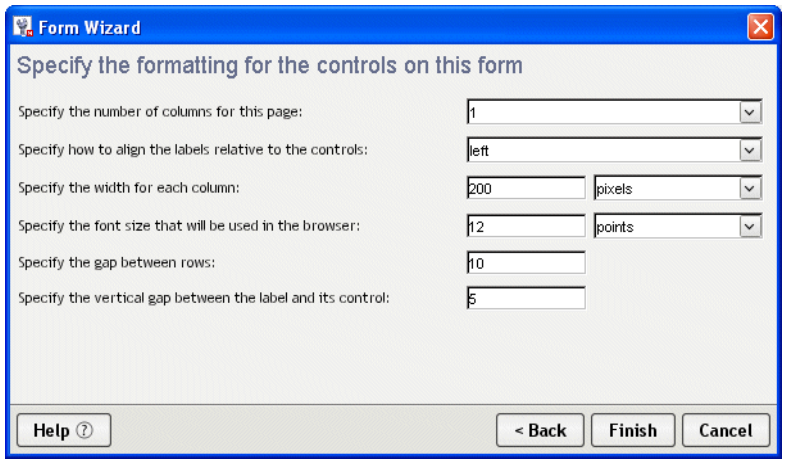
If you chose Schema, you are prompted to specify the root of the instance data defined by this schema on the next pane:

6a Choose the root from the list supplied in the dropdown list.

6b Click **Next**.

7 Click **Next**.

The following wizard panel displays:



8 Specify the following options:

Option	What to do
Specify the number of columns for this page	Specify how many columns of controls the wizard should generate.
Specify how to align the labels relative to the controls	Specify how the labels and the controls should be aligned.
Specify the width for each column	Specify the width and unit for each column. The wizard converts the units to pixels.

Option	What to do
Specify the font size that will be used in the browser	Specify the font size and units to be used.
Specify the gap between rows	Specify the amount of space (in pixels) between rows.
Specify the vertical gap between the label and its control	Specify the amount of space (in pixels) between the label and control.

9 Click **Finish**.

10 When the wizard reports that it has finished creating the form, click **OK**.

What happens When you click Finish, the Form Wizard generates an XHTML file including all of the markup necessary to:

- Build the form's model and instance elements
- Generate and bind default controls for each node of instance data
- Generate a single absolute positioning region containing default controls (and labels) for each component of the instance data
  - If the instance data contains repetitive, homogeneous child elements (indicating repeated data), the wizard generates a repeat block containing default controls
- Lay out the controls based on user input (such as the number of columns and how the label should appear)

About the wizard-generated forms

The generated forms are XForms 1.0-compliant. They reside in an XHTML file. Each form generated by the wizard has the following:

- XML namespace declarations
- XForms model element
- XForms:action elements
- <style> element

XML namespace declarations

XML namespace declarations are located at the top of the file and have this format:

```
xmlns:xforms="http://www.w3.org/2002/xforms"
```

XML namespaces identify the XML language an element belongs to—and how to process them. Some example prefixes include:

xmlns prefix	Description
xforms	Treated as XForms language elements
ev	Treated as XML event language elements

What you do You are not required to change the wizard-added namespaces. You may have to add new namespace declarations depending on your application.

XForms model element

The XForms model element is located within the <head> of the XHTML document. The model element defines the:

- Structure of the data displayed to or entered by the user
- Structure of the data to submit
- Data for initializing form controls
- Rules for validating user data

In the wizard-generated model element you'll see an **id** attribute with a name that includes _wsrp_rewrite_—like this:

```
<xforms:model
   id="model1_wsrp_rewrite_">
```

A model ID is not required (unless there are multiple models); however, the Form Wizard always generates a model ID. The wizard-generated model ID always includes the **wsrp_rewrite** token. At runtime the wsrp_rewrite token is replaced with a unique name. This allows the runtime environment to manage multiple forms and models running within a single page.

What you do You'll use the Model tab to define instance data, submission elements, and model item properties. The runtime environment will manage the replacement of the wsrp_rewrite token.

XForms:action elements

Within the model element, you'll see two predefined XForms <action> elements containing two XForms event handlers. They are default handlers for submit and data validation errors.

They provide user feedback when an error occurs on submission. They are intended to ensure that there are no silent failures for your form. They look like this:

```
<xforms:action ev:event="xforms-invalid" ev:observer="div_wsrp_rewrite_">
<xforms:message level="modal">The data is invalid. Please check it    and try
again.</xforms:message>
</xforms:action>

<xforms:action ev:event="xforms-submit-error">
<xforms:message level="modal">Submission error. Please check the
   data and try again.</xforms:message>
</xforms:action>
```

What you do No action is required, but you can modify them or remove them if you do not want your form to include this functionality.

<style> element

The <style> element contains the CSS definitions for the form and controls.

If you choose to use CSS layout (when generating the form), the styling information for the generated page is contained in CSS styling rules located in a <style> element within the <head> of the XHTML file. You can also store the CSS style rules in an external file referenced from the XHTML file through a link.

NOTE: If you specified not to use CSS styling (when generating the form via the wizard), the generated form will not contain a style node.

The following applies to all XForms controls on a form:

Each control...	Details
Resides in a layout region	A *layout region* is a container for the XForms controls on an XHTML page. It allows you to position controls more precisely. ◆ All layout regions must have the same width. The layout region is defined by a single selector on the form. The selector is: `nv-block-width_wsrp_rewrite_` Each layout region has an individual rule to define height.
Is assigned a class attribute beginning with .nvP followed by a number	The numbers are assigned sequentially. Each control name also contains the token _wsrp_rewrite_ . The token is replaced at runtime and is used to ensure uniqueness of control names when more than one form resides in a single page.
Is styled according to the CSS box model and the CSS pseudo element :: value	There are three types of controls— ◆ Controls with no label, like <repeat> ◆ Controls with an integral label, like <trigger> ◆ Controls whose labels can be positioned separately from the text field, like <input>

	Controls with an integral label are given one selector. For example:
	`.nvP4_wsrp_rewrite_ {`
	`position : absolute;`
	`left : 10px;`
	`top : 91px;`
	`width : 67px;`
	`height : 25px;`

	Controls with separate labels are given three selectors. For example:
	◆ One for a wrapper:
	`.nvP1_wsrp_rewrite_ {`
	`display : block;`
	`}`
	◆ One for the control's label:
	`.nvP1_wsrp_rewrite_ xforms|label {`
	`display : block;`
	`position : absolute;`
	`left : 10px;`
	`top : 1px;`
	`width : 73px;`
	`height : 20px;`
	`}`
	◆ One for the data area of the control:
	`.nvP1_wsrp_rewrite_::value {`
	`display : block;`
	`position : absolute;`
	`left : 88px;`
	`top : 1px;`
	`width : 88px;`
	`height : 20px;`

What you do You can modify the default CSS styling rules by:

- Creating your own style sheets and attaching them to the form
- Adding to or modifying the class styles already defined

📖 For more information, see "Using the CSS Style Manager" on page 102.

Saving forms

➢ **To save a form:**

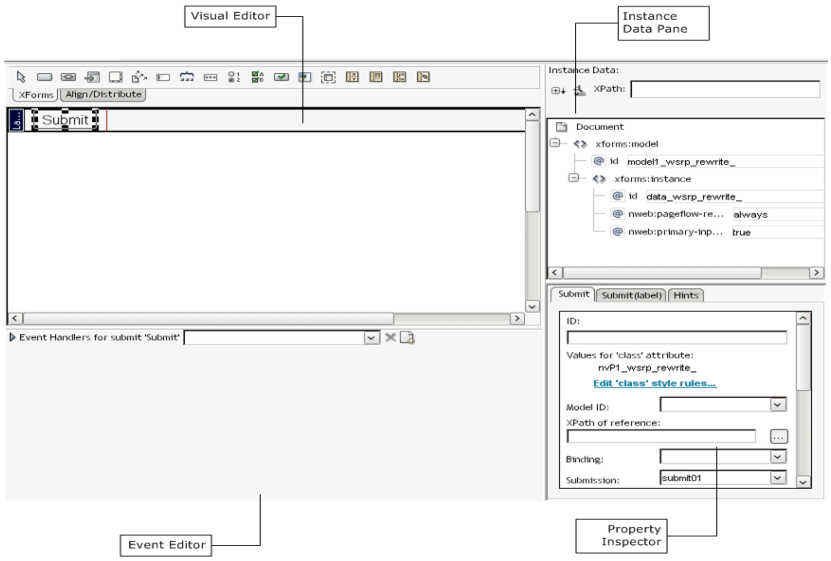
- Click the Save icon from the toolbar.
  OR
- Choose **File>Save**.

Defining the presentation

The Form tab provides the tools to define the user interface. This section describes how to use the Form tab. It includes these topics:

About the Form tab

The Form tab provides a graphical way to create and manipulate the form controls that make up the user interface. The Form tab looks like this:



The Form tab provides:

Tool	Description
Visual Editor	Use the Visual Editor to graphically create and manipulate the controls. The editor includes a tabbed toolbar that includes: ◆ XForms toolbar—the set of controls and blocks that you can drop on your form. ◆ Align/Distribute toolbar—allows you to change the way selected controls are aligned (left/right) or distributed (vertically/horizontally).
Instance Data Pane	Use the Instance Data Pane to bind instance nodes to form controls. You **cannot** use the Instance Data Pane (in the Form tab) to modify the structure of the instance data. Use the Instance Data Pane in the Model tab for those types of functions.
Property Inspector	Use the Property Inspector to manipulate the CSS and data binding properties on the currently selected control. 📖 For more information on using the Property Inspector, see "Setting form control properties" on page 104.
Event Editor	Use the Event Editor to define the events and actions for controls on the form. 📖 For more information, see "Working with model elements" on page 109.

Form tab limitations

Only the form controls in layout regions on the page are editable. You cannot use the Visual Editor to edit the XHTML on the page. The XHTML tags are not expanded to display their content—only the tags are displayed. You cannot insert a form control into an XHTML tag.

Shortcut keys

You can use the following shortcut keys in the Visual Editor:

Keystroke	Description
Ctrl-X	Cut
Ctrl-C	Copy
Ctrl-V	Paste
Delete	Delete
Arrow keys (left, right, up, down)	Moves the selected object 5 pixels in the corresponding direction
Ctrl-arrow key (for example, Ctrl-right arrow)	Moves the selected objects 1 pixel in the corresponding direction
Shift-arrow key (for example, Shift-right arrow)	Stretches the selected object 5 pixels in the corresponding direction
Ctrl-Shift-arrow key (for example, Ctrl-Shift-right arrow)	Stretches the selected objects 1 pixel in the corresponding direction

About form controls

The Form Designer supports all of the XForms controls outlined in the XForms 1.0 specification and several other controls used by the Form Designer to control formatting. The controls include:

Icon	Control	Description
	XForms trigger control	A standard XForms trigger control. For example, a button on a form. Allows user-triggered actions.
	XForms trigger styled as a link	A standard XForms trigger control preconfigured to emulate a link. CSS rules are applied to make the trigger look like a link not a button. This means that: ◆ The trigger contains an <xforms:action> event handler that listens for the DOMActivate event (button press). ◆ The action element contains an <xforms:load> element that loads a new page via an HTTP GET operation. Use the control's property sheet to specify the URL to use for the GET. See also Emulate link, Style as link, Request type, Target URL, Submission in the section on "Setting form control properties" on page 104.
	XForms output	Displays read-only data to the user. This control supports the format property which allows you to specify formatting for certain data types. You apply the formatting in the Property Inspector .
	XForms text area	Allows users to enter freeform, multiline content.
	XForms upload control	Allows users to upload a file from the local file system. **IMPORTANT:** The instance node to which the upload control is bound must be defined as a schema type of base64Binary or you will encounter inconsistent behavior at runtime. For example: `<lastname xsi:type="xsd:base64Binary"/>`
	XForms input	Allows users to enter single-line freeform data. This control supports the format property which allows you to specify formatting for certain data types. You apply the formatting in the Property Inspector . For more information, see "Format" on page 105.
	XForms range	Allows users to select from a sequential range of values.
	XForms secret	Allows users to enter single-line freeform data. The characters are disguised during data entry. Useful for things like passwords.
	XForms Select One	Allows users to select a single item from a set of choices.
	XForms Select Many	Allows users to make more than one selection from a set of choices.
	XForms submit button	A special form of trigger that allows users to submit the contents of the form.
	XHTML image	Displays an XHTML image loaded from the project's resource set. Visible in the Form tab, and View form in browser modes.

Icon	Control	Description
	HTML content box	Read-only display of static HTML content.
	Absolute positioning region	exteNd Director extension. Used for managing layout.
	XForms repeat	Use to display collections of homogeneous data. To manage the repeated elements, add a repeat block to the form, then add the controls representing a single instance of the repeated data within the repeat block. At runtime the processor renders the repeat block once for each data element that the repeat control is bound to. Each instance of the repeated data is processed as a block. Each block is placed below the preceding block. All of the remaining, nonrepetitive content is placed below that. You cannot directly position nonrepeated objects : ◆ Below the repeated elements within the repeat block ◆ To the right of the repeated elements within the repeat block
	XForms switch	Use to perform conditional processing of controls on the form. The switch element allows any number of case elements as children. Each case represents a subform, exactly one of which is rendered at any time by the runtime processor. The case rendered is determined by an action in an event handler not based on the result of a calculation. The Form Designer represents switch elements as a layout region for each case element. You cannot specify the order of the case statements within a switch block—but that is not necessary, since only one will be displayed at a time. The event handler determines how cases are displayed in response to the events that you specify.
	Pageflow link region	exteNd Director extension. At runtime this control is replaced with one or more submit buttons. 📖 For more information, see “Button link” on page 58.

📖 For more information on the properties that you can specify for the controls, see “Setting form control properties” on page 104.

Manipulating controls

After you generate an initial form using the Form Wizard, you can refine it by adding or moving controls. You can use the Form tab for:

- Adding and removing controls
- Moving controls
- Sizing controls
- Aligning controls
- Grouping and ungrouping controls
- Binding controls to data

Adding and removing controls

Adding controls

➢ **To add unbound controls from XForms toolbar:**

1 Click on the control type in the toolbar.

2 Click within the layout area to place the upper-left corner of the control on the form.

➢ **To add unbound controls from the Form Designer menu:**

1 Select **Form Designer>Insert**.

2 Select the control from the popup menu.

3 Click within the layout area to place the upper-left corner of the control on the form.

Removing controls

You cannot remove a control without also removing its label.

➢ **To remove a control:**

1 Click the control.

2 Press the **Delete** key.

OR

3 Select **Edit>Cut**.

Moving controls

➢ **To move a control:**

1 Click the control.

2 Drag it (or use the arrow keys to move it) to the new location.

➢ **To move more than one control:**

1 Select the controls to move by:

- Click one control, then press the Ctrl key and click any other control(s).

  OR

- Click and drag a box around the controls you wish to select—any controls within the box are selected.

2 Drag the set of controls to the new location.

Sizing controls

By default, a control's width and height are unspecified so that they can automatically adjust to their content. You can specify an exact width and height in the two ways described below

NOTE: When you define the width and height, the contents of the control wrap to to accommodate the specified size. The runtime results are not guaranteed.

➢ **To size a form control graphically:**

1 Click the control so that the handles are visible.

2 Size the control by:

- Selecting a handle and dragging it to the desired sized.

  OR

- Selecting a handle and using the arrow keys (Shift-arrow key).

➢ **To size a form control using the Property Inspector:**

1. Select the control.
2. Open the Property Inspector and specify the width and height for the control.

Aligning controls

You can align controls within a layout, repeat, or switch block.

➢ **To align controls:**

1. Select the control to use as a reference, plus the additional controls that you want to align with it.
2. Choose the alignment you want (from the Align toolbar) to apply to the selected objects.

   OR
3. Choose **Form Designer>Align Distribute selected objects**.

Grouping and ungrouping controls

You may want to create groups of controls, because a group can:

- Provide a hint to the client side XForms renderer, so that related controls can appear together on limited capability devices, such as cell phones and PDAs.
- Simplify the XPath expressions within a group by establishing a context for the controls in the group.
- Provide a container for the controls to which you can apply CSS styles within the group via inheritance.

Rules for grouping controls

- A control can belong to more than one group.
- Groups must be strictly nested—that is, the entirety of a group must be contained by a higher-level container such as group, repeat, or switch/case blocks.
- The entirety of a control (not just the label, for example) belongs to a group.
- Be careful when using groups. Grouping a set of controls in the Form Designer does not lock the controls together spatially. You cannot ungroup and regroup a set of controls without losing information (such as the instance data context, or style information attached to the group element).

Most of these rules are required because of the hierarchical nature of XML.

➢ **To create a group:**

1. Select the controls you want to group.
2. Select **Add to Group** from the Form Designer menu or by right-clicking and selecting it from the popup menu.
3. Choose **New Group**.
4. Enter a name for the group.

   The Form Designer creates the new group, and it becomes the current selection. The Property Inspector displays properties for the selected group. When the group is selected, it can be dragged around the page to reposition the controls within the group.

What happens when you group controls

When you create a group:

- The XML is reorganized to create a new group element whose parent is the parent of the control that is the primary selection.

- The selected controls become children of the new group element.
- The nodeset binding for the group node is set to the instance data node that is the “lowest” ancestor to the instance data nodes bound to the selected controls
- The XPath bindings for the controls in the new group are adjusted to become relative to that node.
- Binding specifications using the bind attribute are not adjusted.

➢ **To add a control to a group:**

1. Select the control to add.
2. Select **Form Designer>Group>Add to Group** (or right-click to display a popup menu).
3. Select the name of the group to add the control to.

   The selection does not change to the entire group, in case there are additional commands to be performed on the selected control. The XML element for the control is moved under the group element representing the group, and the instance data binding XPath is adjusted to be relative to the node bound to the group element.

➢ **To remove controls from a group:**

1. Select the control(s) to remove.
2. Right-click and select **Remove from group** (or select it from the Form Designer menu).
3. Choose the group from which the element should be removed.

   The XML element(s) representing the control(s) are detached from the group element and made siblings of it; the instance data XPath expressions are adjusted to absolute expressions. If no controls remain in the group, the group element is removed from the document.

➢ **To remove a group:**

- To remove a group (but not its associated controls) from a form, you can:
  - Select **Form Designer>Group>Remove Group**.

What happens when you remove a group

When you remove a group:

- The XML elements representing the controls in the group are detached from the group element and made siblings of it.
- The group node is deleted.
- The instance data XPath expressions for the controls are adjusted to absolute expressions.

Applying styles to controls

The Form Designer provides default styles (based on standard portlet CSS class definitions) to implement color, sizing, and fonts used in the form’s presentation. The default class attributes for the styles are defined in wsrp-classification.xml (located in the \Common\Resources\CSSClassifications directory). At runtime, the various portal theme files define the styles associated with these class attributes, and with some internal Novell class attributes.

Changing CSS Classifications

The wizard generates class attributes for each of the controls on a form. For example, an input field on a form could be given a name like:

```
class=”nvP1_wsrp_rewrite_ portlet-form-input-field”
```

- The **nvp1** class value references a unique selector in the internal style node that specifies the absolute positioning rules for this element.

- The **wsrp_rewrite** is a token that is replaced at runtime to ensure unique names for all of the controls on a page in a multiportlet page.
- The **portlet-form-input-field** is the class name that references a formatting rule in an external style sheet (normally a portal theme file).

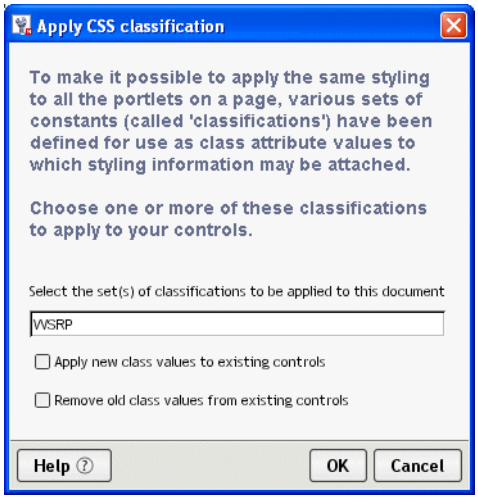
These classes are defined in WSRP-classification.xml in the Common\Resources\CSSClassifications directory. This file maps the class names to particular control types. These class names are added by the wizard if you check the Standard Portlet CSS Classes check box (on the first page of the wizard).

➢ **To apply CSS Classifications to a file that was not generated with these values:**

1 Open the form in the Form tab.

2 Choose **Form Designer>Set CSS Classification**.

The Apply CSS classifications dialog displays:



3 Complete the panel as follows:

Field	Description
Select the set(s) of classifications to be applied to this document	You must select WSRP. **NOTE:** Further sets of classifications may be made available in the future.
Apply new class values to existing controls	Check this box to have the new class values added to existing controls, as well as to new controls that get created later.

Remove old class values from existing controls	Check this box to strip all of the classification-defined class values from the controls on the form.
	For example, a control with the attributes:
	`class="nvP1_wsrp_rewrite_ portlet-form-input-field"`
	will become:
	`class="nvP1_wsrp_rewrite_ "`
	If you check this box, you will lose any formatting associated with the class values that were removed.

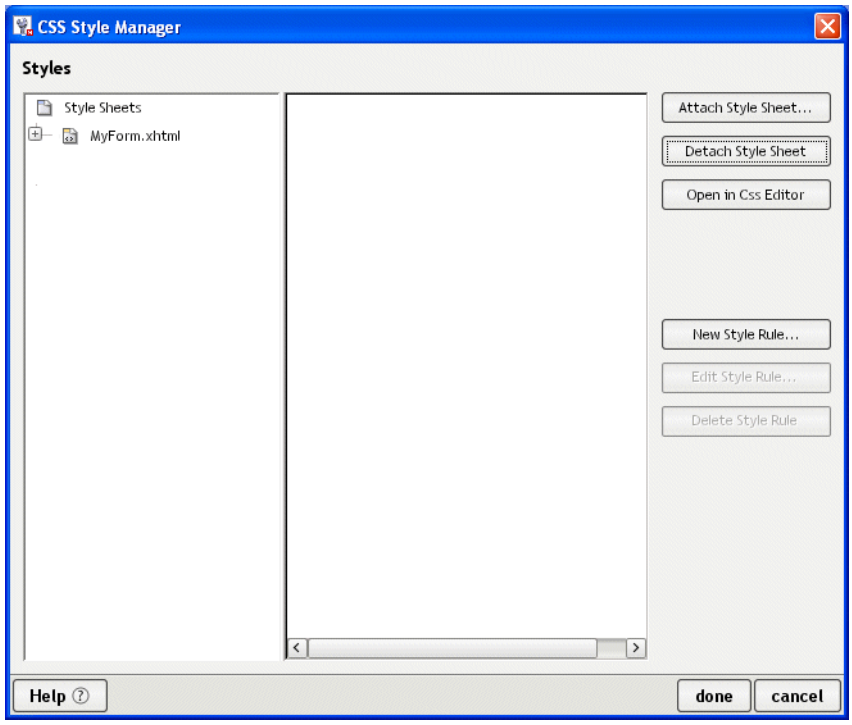
4 Click **OK**.

Using the CSS Style Manager

You can use the CSS Style Manager to edit any internal or external CSS style sheets associated with your form from within the Form Designer. You can use it to specify CSS properties like background color, text color, and font size associated with specific controls and labels. (The Form Designer directly handles control and label positioning; you cannot use the CSS Style Manager for this function.)

➢ **To open the CSS Style Manager:**

1 With a form open, choose **Form Designer>Style Manager**.



NOTE: Pressing cancel on this main dialog does not cancel completed actions.

You'll use the CSS Editor to create, edit, and delete style rules for both internal and external style sheets.

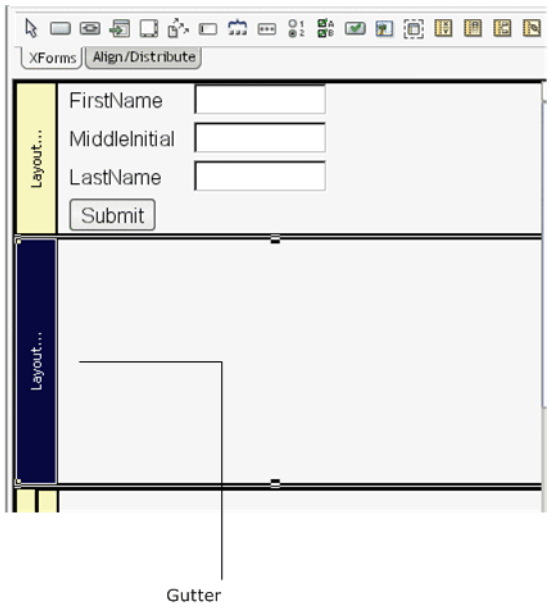
📖 For more information, see the chapter on the CSS Editor in *Utility Tools*.

Working with layout regions

To allow you more precise control of the layout of the controls on a form, the Form Designer requires you to place form controls within a *layout region*. A layout region is a container for the XForms controls within an XHTML page. Like other form controls, you can add, remove, size, and set properties on layout regions. The following controls can act as layout regions:

- Absolute positioning region
- XForms repeat control
- XForms switch/case control
- Pageflow link region

Layout regions are identified with a gutter in the far left of the Visual Editor. :



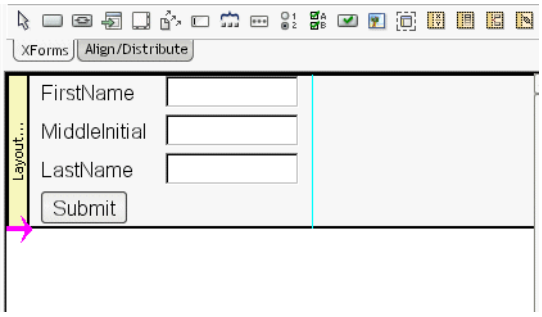
Click the gutter to select the control for setting layout region properties, resizing, or deleting.

➢ **To add a layout region to a form:**

1 Choose the layout control to add from the XForms toolbar.

2 Position the cursor to the location where you want to add the control.

The cursor displays as a pink arrow.



3 Click to place the layout region.

The layout region is added to the form.

➢ **To remove a layout region:**

1 Select the layout region (it is selected when the gutter label is highlighted).

2 Right-click and choose **Delete**.

➢ **To resize a layout region:**

1 Select the layout region to resize.

2 Grab the handles of the layout region and drag the box to the required size.

➢ **To create nested repeat blocks:**

- Click the **Insert XForms Repeat** icon in the XForms toolbar.

  Click in the white space of an existing repeat block in the Visual Editor (click away from the block's boundary—clicking on or near the lower boundary of the repeat block adds the new repeat after, not within, the existing block).

The Form Designer adds an empty repeat group to the form. It looks like this:

➢ **To create a switch block:**

1 Click the **Insert XForms Switch** icon in the XForms toolbar.

2 Move the pointer to the white space at the bottom of the bottommost block.

The Add New Cases to Switch dialog displays.

3 Enter two or more case IDs separated by spaces.

You'll use the case IDs to program the behavior in the Event Editor.

4 Click **OK**.

The The Form Designer adds a switch block with a case block for each case id you entered.

5 Add controls to the case layout regions, as described in "Adding and removing controls" on page 98.

TIP: You can add or remove cases by selecting the switch box, right-clicking and choosing the action from the popup menu.

Setting form control properties

Properties are attributes that you can set at design time for a particular control. You specify property values in the Property Inspector. See the table below for properties specific to the exteNd Director extensions (such as the Pageflow link region) or conveniences (such as the Alert literal text). For all other definitions, see the XForms specification.

Property name	Description	Applies to
Alert literal text	The text displayed in an alert box. 📖 See "Informing users of validation errors" on page 118	All controls for which model item properties can be defined
Button height	Specifies the height of buttons added to the Pageflow link region at runtime. The default is 25.	Pageflow link region
Button layout	Specifies the layout of the buttons added to the Pageflow link region at runtime. The default is Horizontal.	Pageflow link region
Button spacing	Specifies the spacing between the buttons added to the Pageflow link region at runtime. The default is 5 pixels.	Pageflow link region

Property name	Description	Applies to
Button width	Specifies the width of the buttons added to the Pageflow link region at runtime.	Pageflow link region
Dynamic choices	See "Populating the Select controls" on page 107	XForms Select Many
Edit class style rules	Launches a dialog that lets you: ◆ Modify the CSS class styles on the currently selected control ◆ Create new or edit existing styles (by launching the CSS Editor)	All controls for which CSS styling applies
Emulate link	Defines a trigger control's appearance. **Checked**—Trigger is styled as a link. Adds an <xforms:action> event handler that listens for the DOMActivate event (button press). The action element contains an <xforms:load> element that loads a new page via an HTTP GET operation (and thus the Request type is set to Get). There is no default URL to use for the GET; you have to add that via Target URL. **UnChecked**—Trigger is styled as a button.	XForms trigger control, XForms trigger styled as a link, Pageflow link region
Field type	Sets the data type that the control assumes for the instance item it is bound to. Use this in conjunction with the Format property. Make sure that you specify a field type that is valid for the data type of the field. When there is a conflict between these two types: ◆ The instance node the form control is bound to becomes invalid. ◆ The event xforms-invalid is dispatched to form controls bound to that node. ◆ The form control becomes invalid and thus the CSS styles with the :invalid pseudoclass apply. ◆ If the invalid instance node is part of the document fragment being submitted, the submit fails. See also Format	All controls that can be bound to data
Format	Specifies how to format the data when displaying it. Valid formats are: Number, currency, percent, ###,###.##, #####0.##, #####.00 See also Field type	All controls that can be bound to data
Hint literal text	The text displayed in a tooltip.	All visible controls
ID	A unique identifier for the control.	All controls
Label	The text displayed for the control's label.	All controls that can have a label
Model ID	The ID of the model associated with the Form control.	All controls that can be data bound.

Property name	Description	Applies to
Request type	**Get**—Uses the <xforms:load> action. Requires a Target URL property. **Post**—Uses the <xforms:send> action. Requires a Submission property.	XForms trigger control, XForms trigger styled as a link, Pageflow link region
Source	Specifies the name of the source file containing the XHTML image to insert.	XHTML image
Style as link	**Checked**—The XForms Trigger control is styled as an HTML link. Adds a CSS class selector (**nv-link-style**) to the trigger's class attribute. This CSS selector styles the trigger to look like the default appearance of the HTML <a> tag. This selector rule is added to the page's <style> node so you can modify it to look the way you want. All triggers marked to emulate links share this same style. **Unchecked**—Removes the selector name from the trigger's class attribute, but does not delete the selector rule from the <style> node.	XForms trigger control, XForms trigger styled as a link, Pageflow link region
Submission	Specifies the value of <xforms:send>'s submission attribute. (This is the ID of an <xforms:submission> in the <xforms:model>.)	XForms trigger control, XForms trigger styled as a link, Pageflow link region
Target URL	Specifies the value of <xforms:load>'s resource attribute. (The URL that a GET is sent to in order to load the new page.) This URL automatically gets marked up for URL rewriting in the portlet context, so specifying a relative URL here will get back to your portlet.)	XForms trigger control, XForms trigger styled as a link, Pageflow link region

Binding controls to data

The Form Designer makes it easy to bind controls to a single node or a node set. In many cases you won't need to take any action to bind a control to data. For example:

- The Form Wizard automatically creates data-bound controls
- When you drag a data node from the Instance Data Pane onto a control in the Visual Editor, the Form Designer generates the data bindings automatically

If you want to modify the generated bindings, or if you want to bind controls that you've added to the form from the toolbar, you can use the Property Inspector to specify the binding attributes. You can also set up a <bind> in the Model tab, then set the bind property to the <bind> element's ID. See "Binding elements to controls" on page 118.

NOTE: The following procedure generates a binding via a <ref> attribute on the control. It's possible to specify either a <ref> or a <bind>. When you specify one, the Form Designer removes the other. If you enter both via the Source tab, the <bind> takes precedence.

➢ **To bind a control to a single node via an XPath:**

1. In the Visual Editor, select the control you want to bind.
2. Using the Property Inspector, specify the model ID (when there is more than one model element for the form).

3 Specify the XPath by:

- Typing the XPath in the text box of Reference (generates a <ref> attribute)

  OR

- Launching the XPath Navigator (by clicking the ellipsis next to the XPath text box)

3a In the XPath Navigator dialog, locate the node you want to bind to and select it.

📖 For more information on using the XPath Navigator, see the chapter on scoped paths and XPaths in *Developing exteNd Director Applications*.

3b Click **OK** to return to the Form Designer.

➢ **To bind a control to a node set:**

NOTE: <repeat> elements require that you bind to a node set.

1 In the Visual Editor, select the control you want to bind.

2 Using the Property Inspector, specify the **model ID**.

2a In the XPath Navigator dialog, locate the node you want to bind to and select it.

📖 For more information on using the XPath Navigator, see the chapter on scoped paths and XPaths in *Developing exteNd Director Applications*.

2b Click **OK** to return to the Form Designer.

NOTE: If a <ref> resolves to multiple nodes, the first node is used.

Populating the Select controls

There are two ways to provide a list of values for the Select controls:

Method	Enables you to	When to use
Statically	Specify the list values while you are designing your form.	Use this method when the list is relatively short, you know what the values are, and the values won't change.
Dynamically	Specify the list values come from a nodeset.	Use this method when the information is located in an XML file. When you do not know what the list items will be or the list changes frequently.

List values Lists have two types of values:

- A label (for display)
- A value (for storage)

The user sees the label, but the associated value is the value written to the instance node. This allows you to display user-recognizable text while storing keys or other types of codes in the instance node. For example, if you entered the words apples, oranges, and pears, these words would appear in the display list. However, selecting apples might return the value 1, oranges could return 2, and so on.

Instance elements A Select control is usually associated with two instance elements:

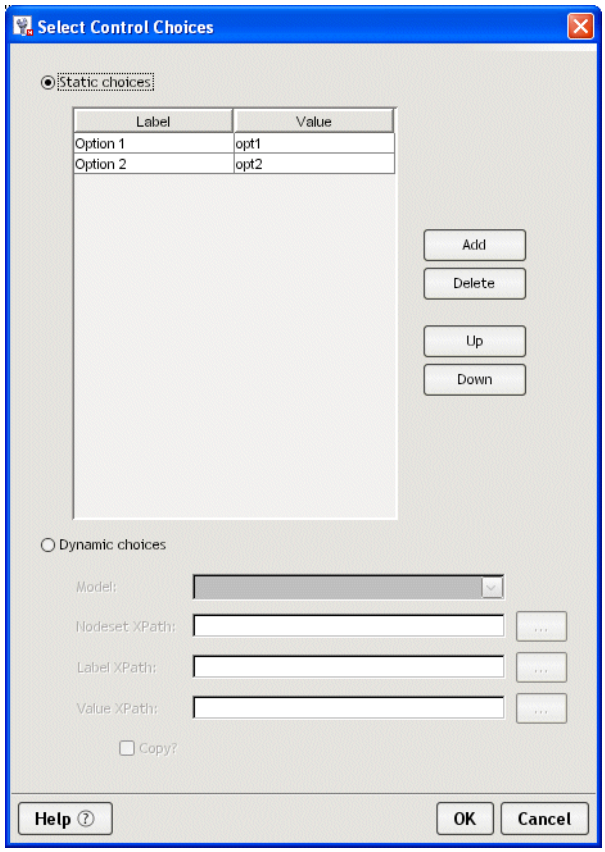
- One containing the display values
- One containing the element to which the value is written

➢ **To load the list statically:**

1 Highlight the **Select** control and access the property sheet.

2 Choose the **Edit Select Choices** link.

The Select Control Choices dialog displays.



3 Click the **Static Choices** radio button.

4 The dialog enables the Label and Value text boxes:

- To change the predefined Label and Value choices, double-click within the text box and change the text.
- To add a new choice, click **Add**, and place the cursor within the added text box and add the Label and Values.

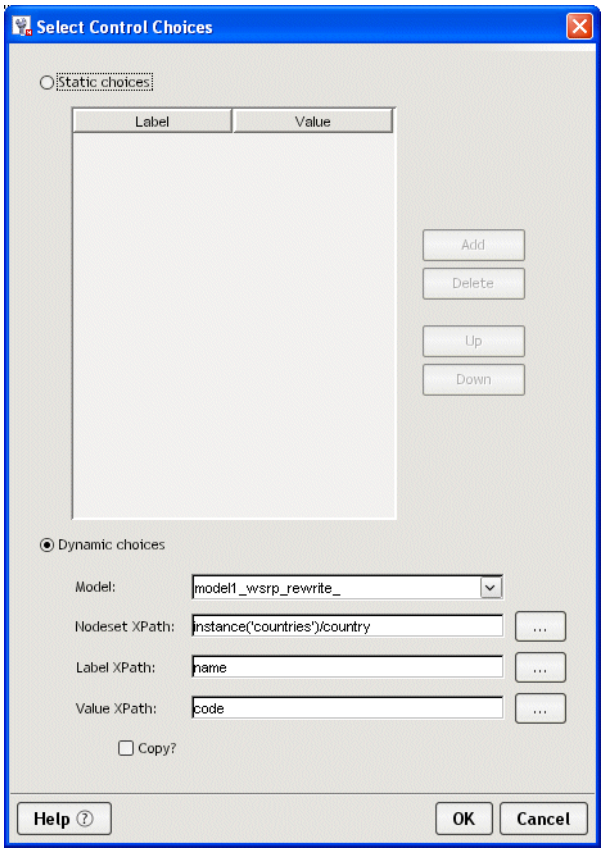
5 When you are done, click **OK**.

➢ **To load the list dynamically:**

1 Highlight the **Select** control and access its property sheet.

2 Choose the **Edit Select Choices** link.

The Select Control Choices dialog displays:



3 Click the **Dynamic choices** radio button.

4 Choose the model from the dropdown list box.

5 Click the ellipsis next to the Nodeset XPath to access the XPath Navigator to choose a nodeset.

6 Click the ellipsis beside Label XPath to access the XPath Navigator to the display value.

7 Click the ellipsis beside the Value XPath to access the XPath Navigator to choose storage value. This value is written to the instance node.

8 When you are done, click **OK**.

Working with model elements

The XForms model element defines the structure of the XML data available to the form. It defines the:

- Structure of the data displayed to or entered by the user
- Structure of the data to submit
- Data for initializing form controls (instance data)
- Rules for validating user data

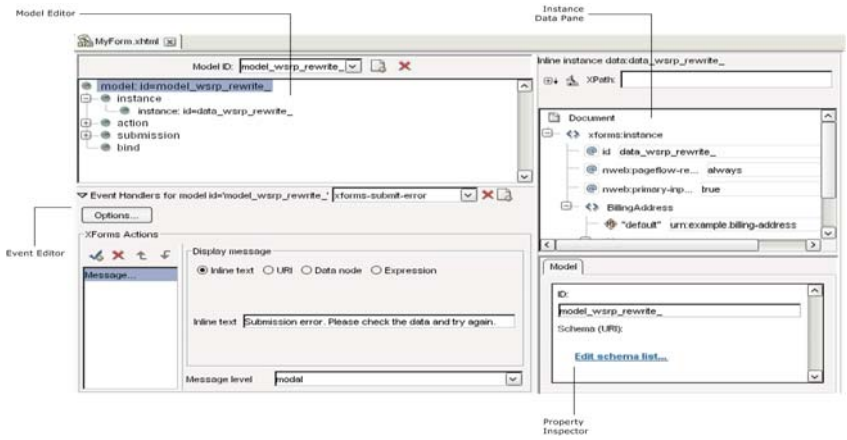
The Model tab includes these elements:

Element	Description	For more information, see
Model	The model root	"Specifying model elements" on page 110

Element	Description	For more information, see
instance	Points to or contains the data used to initialize the form	“Specifying instance elements” on page 112
action	Defines event handlers and actions that can be accessed from any part of the form	“Specifying actions” on page 114
submission	Defines the set of data to submit and how to submit it	“Specifying submission elements” on page 114
bind	Defines properties (called model item properties) of the instance data—like readonly, relevant, calculations, and indirect binding, and so on	“Specifying Bind elements” on page 116

About the Model tab

The Model tab provides a graphical way to define the elements that comprise the form’s model. The Model tab looks like this:



The Model tab provides:

Tool	Description
Model Editor	Use to add and remove elements.
Instance Data Pane	Use the Instance Data Pane to modify the structure of the instance data. For more information on using this tool, see XML Editor in the online *Utility Tools*.
Property Inspector	Use to create and modify attributes on the selected model element.
Event Editor	Use to define the events and actions on the selected model element.

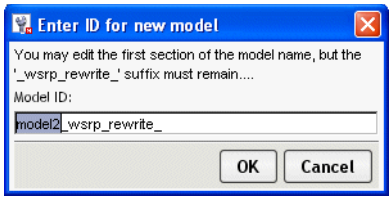
Specifying model elements

By default, the wizard-generated forms include a model.

➢ **To add a model element:**

1 With the form open in the Model tab, click **Add** (at the top of the Model Editor).

The following dialog is displayed.



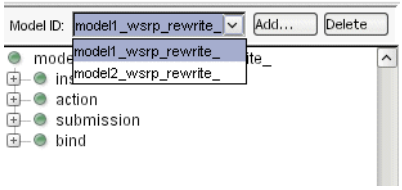
You'll see that wsrp_rewrite_ is appended to the model ID name. This is a placeholder that is recommended. At runtime the wsrp_rewrite_ is replaced with a unique ID to ensure that no naming conflicts occur among other forms or portlets on the same page.

2 Name the model and click **OK**.

The Form Designer creates a new, empty model tree and displays it in the Model Editor.

➢ **To remove a model element:**

1 With the form open in the Model tab, select the model element you want to delete from the dropdown list box.



2 Click **Delete**.

➢ **Specifying model properties:**

1 Select the model in the Model Editor.

2 Access the Property Inspector and complete the properties as follows:

Property	Description
ID	Specifies a unique identifier for a model. A model ID is not required (unless there are multiple models), but the Form Wizard always generates a model ID and uses the token wsrp_rewrite. It is good practice to use an ID because at runtime there might be multiple portlets with multiple XForms on a single page.
Schema URI	Specifies a list of external schema documents that are needed to describe the structure of the instance data and allow it to be validated. To add schema documents: **1** Click **Edit schema list**. (The Edit Schema File List dialog displays). **2** Click **Add**. **2a** In the Select Schema File dialog, specify the name and location of the schema file. **2b** Click **OK**. **3** Click **OK**.

For information about adding events, see "Customizing event handlers" on page 130.

NOTE: Some element types only allow you to add elements to the root (like Instance). Others allow you to nest elements within other elements (like bind). You can determine which elements can be nested by selecting an element (not the root) and right-clicking to see if **Add item** is offered on the menu.

Specifying instance elements

➢ **To set instance data properties:**

1 Select an instance element (not the root).

The instance element's properties are displayed in the Property Inspector. They include:

Property	Description
ID	A unique identifier for the instance data. This is necessary only when there are multiple instance data nodes in a single document. It is good practice to always allow the Form Designer to add _wsrp_rewrite_ because at runtime there might be multiple portlets with multiple XForms on a single page.
Use pageflow data at runtime?	Specifies how instance data should be handled at runtime (once the form is incorporated within a pageflow). Options are: ◆ always (the default)—Always replace the data at runtime. If no replacement data is available, a runtime error is generated. ◆ if-available—Replace only if new data is available at runtime; otherwise use the design-time data. ◆ delete-if-not-available—Use new data if it is available at runtime. Don't ever use the design-time data—delete it. ◆ never—Use the design-time data alone. **IMPORTANT:** When you incorporate a form into a pageflow (as an XForms activity), you'll be able to specify a set of scoped paths (using the Property Inspector for the activity), indicating the replacement data for each of the replaceable <xforms:instance> nodes. (These are the instance nodes whose **Use pageflow data at runtime** is set to always, if-available, or delete-if-not-available.)
Is primary instance data?	Specifies whether data for an <xforms:instance> node is treated as the **primary instance data**. The primary instance data is the input data to the page rendering's XSLT transformation; it is the default context for use in XSLT expressions in the page. All other input data is **secondary input data** and can be accessed only via variable references in XSLT expressions. 📖 For more information, see "About runtime replacement of instance data" on page 113.
Is inline?	**Check** this box when data is contained within the model element of the form. When this is checked, you can use the instance data generated by the wizard or you can import the instance data from a file located within the project's resource set. See Import instance data from file (below). **Uncheck** this box when you want the data to be referenced from an external file. See Source (URI) (below).

Property	Description
Import instance data from file	To import instance data: **1** Click the ellipsis button. **2** In the Import File into Instance Node dialog, **2a** Type the name of the file to import. or **2b** Click **Import**, navigate to the file to import, and click **Open**. **3** Click **OK**.
Source (URI)	When the data is not inline, you can specify the location of the data using a URI. **NOTE:** The URI must reference a location within the current project's resource set. To add a link to the instance data: **1** Click the ellipsis next to the Source URI text box. **2** In the Select File dialog, choose the file and click **OK**.

About runtime replacement of instance data

At runtime, the form's data is replaced as specified by the **Use pageflow data at runtime?** property. The data is replaced for each form (which runs as a portlet) separately, and it happens before aggregation into the portal page. Additionally, the data corresponding to one or more input documents may be made available, according to the following rules:

Rule	Description
No replaceable <xforms:instance> element is specified as the primary instance data.	◆ If there is only one instance element, that one instance element is treated as the primary instance data. The primary input document is mapped to the primary instance data.
	◆ If there is more than one default <xforms:instance> element in the page, one instance element must be defined as the primary instance data—and: ◆ All other replaceable <xforms:instance> elements use secondary input data. ◆ All secondary input data is made available to XSLT expressions in the page via XSLT variables. XSLT allows only a single input document, so other documents must be made accessible via the XSLT variable mechanism. ◆ The variable identifiers are equal to the ID of the corresponding <xforms:instance> elements. (You use a dollar sign in XSLT to access a variable in an XPath expression; thus if you had <xforms:instance id="foo"> you'd access its runtime data via something like <xsl:value-of select="$foo/a/b/c"/>.)

📖 For more information on adding events to the instance element, see "Working with events and actions" on page 118.

Specifying actions

The Form Designer provides multiple ways for defining actions within your form:

Method	Description
The action node of the Model tab	Use the action node of the Model tab as a place to create one or more actions (with unique IDs) and then reference them elsewhere in your form.
The Event Editor in the Form tab	Use the Event Editor launched from the Form tab to create actions for your form controls 📖 For more information, see "Working with events and actions" on page 118.

The remainder of this section describes how to specify actions in the Model tab.

➢ **To create an action/event handler:**

1 From the Model tab, select the top-level **action** node from the model tree.

2 Right-click and choose **Add item**.

3 Navigate to the Property Inspector and supply the following properties:

Property	What to specify
ID	Provide a unique name for the action/event handler.
Event name	Choose the event from the dropdown list.
Observer	(Optional) Choose an element as the observer.
Target	(Optional) Choose an element as the target of the action.

4 Right-click and choose the action you want performed for the event.

5 Navigate to the Property Inspector to define the specification for the selected action.

Specifying submission elements

The submission element defines:

- The structure of the data to submit
- Where to submit it
- How to submit it

➢ **To create a submission element:**

1 From the Model tab, select the top-level **submission** node from the model tree.

2 Right-click and choose **Add item**.

3 Navigate to the Property Inspector and supply the following properties:

Property	Description
ID	Specifies a unique ID for the submission element.
Binding ID	Specifies the ID of a Bind element. Choose the ID of the bind element that specifies the node(s) to submit.
XPath of reference	An XPath specifying the node(s) to submit.
Action (URI)	Specifies the URI to the location where the submission is sent.

Property	Description
Method	Specifies how to do the submission. Values: ◆ Post ◆ Get ◆ Put ◆ form-data-post ◆ multipart-post (treated like a Post) ◆ urlencoded-post
Replace	Specifies what the processor should do with the document returned after the submission. Values: ◆ all (the default) ◆ instance ◆ none
Separator	Specifies what separator character to use on url-encoded serialization. Values: ◆ ; ◆ &
Indent	For application/xml serialization only. Specifies whether to insert white space. Values: ◆ True ◆ False
Standalone	For application/xml serialization only. Specifies whether to include a declaration. Values: ◆ True ◆ False
Omit XML declaration	For application/xml serialization only. Specifies whether to include an XML declaration. Values: ◆ True ◆ False
Encoding	For application/xml serialization only. Specifies the type of encoding to use.

Property	Description
Edit namespace list	For application/xml serialization only. Specifies the namespaces to include in the serialized XML. Not specifying any namespaces is the same as specifying all. To edit the namespace list: **1** Click **edit namespace list**. **2** Select the namespaces you want to include. Use Shift to select a contiguous group and Ctrl-Shift to select multiple non-contiguous items. **3** Click **OK**.
Mediatype	Specifies the Internet media type for the serialized instance data.
Version	For application/xml serialization only.
CDATA section elements	For application/xml serialization only. To add CDATA elements: **1** Click **Edit CDATA element list**. **2** In the Edit CDATA item list dialog, click **Add**. **3** Type the name in the Add List dialog. **4** Click **OK**. **5** Click **OK**.

For more information on adding events to the submission element, see "Working with events and actions" on page 118.

Specifying Bind elements

Use the Bind element to set rules on the instance data. For example, you can define fields as required, enabled, or disabled depending on selections that the user makes, and so on. You can also use a bind element as an indirect way of specifying the binding for a control; instead of using <ref> or <nodeset>, use the bind=bindingID attribute on the control. Once you have defined the bind elements in the Model tab, you can return to the Form tab and associate them with form controls to create a UI Binding expression.

➢ **To create a bind element:**

1 Select **bind**.
2 Right-click and select **Add item**.

➢ **To delete a bind element:**

1 Select the bind element.
2 Right-click and select **Delete item**.

➢ **To set properties on a bind element:**

1 Select the bind element.
2 Access the Property Inspector and complete the properties as follows:

Property	Description
ID	A unique ID for the bind element.

Property	Description
XPath of Nodeset	The node(s) in the bind.
Model Item Properties	Lets you define the model item properties to apply to the nodes defined in the XPath of Nodeset above. For more information, see Specifying model item properties below.
Type	Specifies the XML schema data type for the associated node. The Form Designer includes a convenience that allows you to specify formatting for certain data types. Only the data types shown in bold support this formatting. You apply the formatting in the Form tab Property Inspector for the control bound to such a node. For more information, see "Format" on page 105

Specifying model item properties

You can specify these Model Item properties like this:

Model item property	Description
Readonly	Users are not allowed to change the data. Any form controls bound to a read-only node are not enabled. To make a node readonly: **1** In the Property Inspector, choose **readonly**. **2** Create an XPath expression that evaluates to a boolean, or change the value to true().
Required	Users are required to supply a value. Any form controls bound to a required node will generate a submit error if a value is not supplied. To make a node required: **1** In the Property Inspector, choose **required**. **2** Create an XPath expression that evaluates to a boolean, or change the value to true().
Relevant	Specifies whether a node is visible. Form controls bound to nonrelevant nodes are disabled or not visible. If you make a nonrelevant node required, the required property is ignored. To make a node nonrelevant: **1** In the Property Inspector, choose **relevant**. **2** Create an XPath expression that evaluates to a boolean, or change the value to false().
Calculate	Specifies a calculation that defines the value of the node. To create a calculation on a node: **1** In the Property Inspector, choose **Calculate**. **2** In the XPath Navigator, construct the calculation. For more information, see the chapter on scoped paths and XPaths in *Developing exteNd Director Applications*.
Constraint	Specifies a boolean expression that when false causes the associated model item to be regarded as valid. (The converse is not necessarily true.)

Informing users of validation errors

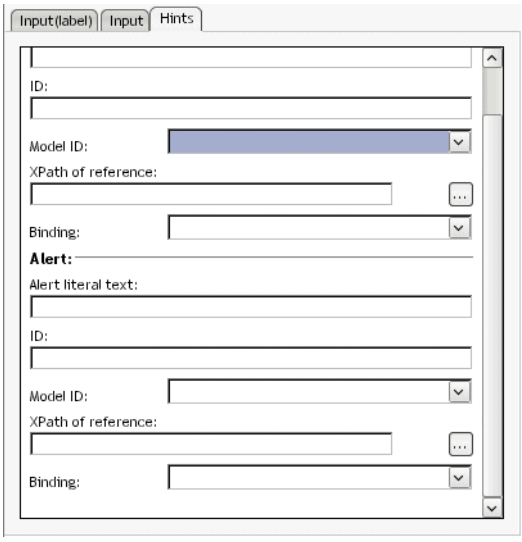
The Form Designer makes it easy to notify users that a control (or the form) has failed one of the model item property validation tests.

➢ **To notify users of an error:**

1 When you've defined the model item properties, access the Form tab.

2 Select the control bound to the node for which the model item property is defined.

3 In the Property Inspector for the control, access the Hints tab.



4 Choose the Binding associated with the model item property definition from the dropdown.

5 Type a message in the **Alert literal text** to display to the user when a failure is encountered.

Binding elements to controls

When you've defined the bind elements in the Model tab, you can return to the Form tab and associate them with form controls to create a UI Binding expression.

Working with events and actions

XForms supports and extends the DOM Level 2 event model which is based on the XML event model.

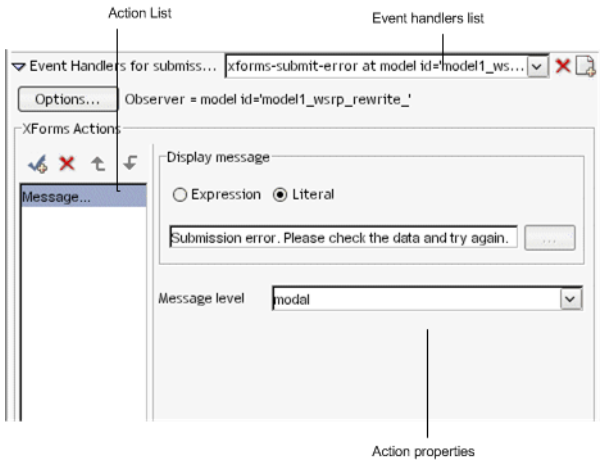
You use the Event Editor to create event handlers and define XForms Actions.

This section includes the following information:

- About the Event Editor
- XForms Actions Reference
- Customizing event handlers

About the Event Editor

The Event Editor is available on both the Form tab and the Model tab. It looks like this:



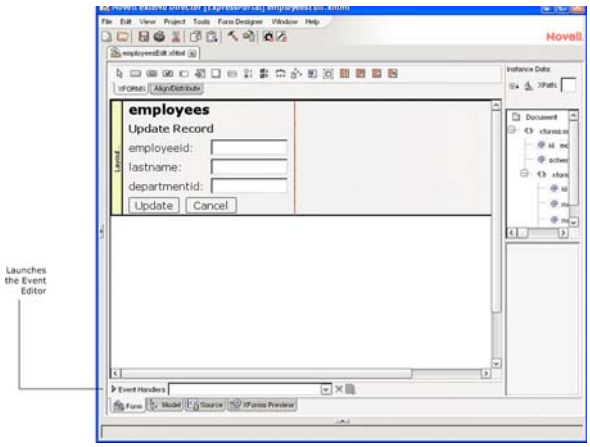
The Event Editor is enabled when you select an item that allows event handlers such as a form control, a submission element, or a bind.

The Event Editor supports all XForms events and actions, but not all events are presented as choices when creating an event handler. If an event is not presented as a choice, you can type it in the choice box or use the Source tab to define it.

➢ **To launch the Event Editor:**

1 Navigate to the bottom of Form Designer (in the Form or Model tabs).

2 Click the icon next to the Event Handlers label. The following graphic shows the location of the icon in the Form tab.



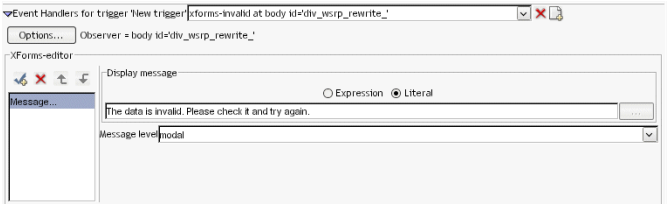
The Event Editor opens.

3 Select a form control or model item for which you want to define an event.

The Event Editor is now available to edit existing or create new Event Handlers.

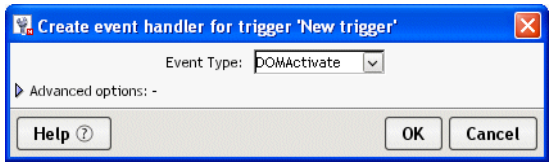
➢ **To create an event handler:**

1 Select the element that you want to define the event handler for.



2 Click the Create a new event handler icon.

The Create Event Handler for *Element* dialog displays:



3 Choose an event from the Event Type dropdown, or type an event name (if it is not listed).

3a If you want to further customize the event handler by adding observers, and default actions, click **Advanced options** and see "To customize event dispatching:" on page 130.

4 Click **OK**.

The Event Editor now allows you to choose XForms Actions to respond to the event handler (described next).

NOTE: You can't put an event handler on the instance element because instance nodes are restricted to a single child element. When you use the Event Editor to add an event handler to the instance element, it will put it on the model.

➢ **To specify XForms Actions for an event handler:**

1 After creating an event handler, click the New Action icon.

2 Select an XForms Action from the popup list.

The selected action and it's properties are displayed.

3 Complete the properties.

4 Save the form.

The Event Editor generates the event handler and XForms Action defined by the properties you specified.

For more information about Action properties, see the XForms specification.

➢ **To delete an action:**

1 Select the action from the Action List.

2 Click the delete icon.

➢ **To delete an event handler:**

1 Select the event handler from the Event Handler list.

2 Click the delete icon.

XForms Actions Reference

Click an action to display complete information.

Action	Action
Delete	Reset model

Action	Action
Dispatch	Send instance data
Insert	Set Focus
Load link	Set index in repeat
Message	Set value
Rebuild, Recalculate, Refresh, and Revalidate model	Toggle select case in switch

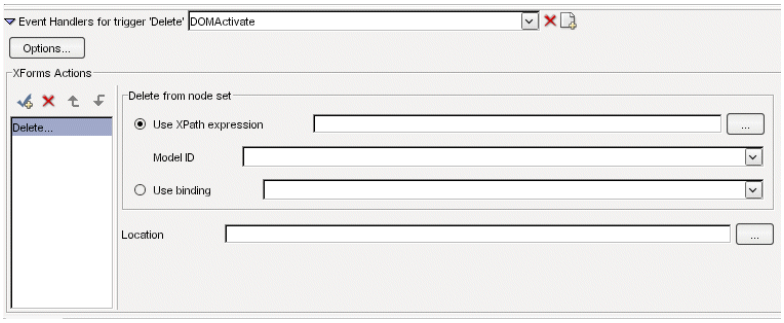
Delete

Description

Deletes a specific node of repeated instance data. Typically, the nodeset is bound to a repeat block.

Attributes

To construct the action, use the Event Editor to define it's attributes.



Complete the attributes as follows:

Attribute	What to do	Notes
Use XPath expression	Choose **Use XPath expression** to specify an XPath expression that identifies the data node to delete.	You can type the expression in the choice box, or click the ellipsis to launch the XPath Navigator. The Event Editor constructs an <xforms:action> with a nodeset definition.
Model ID	Optional. Choose a Model ID from the choice box.	A Model ID is only needed when: ◆ You choose **Use XPath expression**. ◆ The form references multiple models containing instance data elements of the same name.
Use binding	Choose **Use binding** to specify an existing bind expression that identifies the data node to delete, then choose a binding ID from the choice box.	The choice box is only populated if you've defined a binding element **and** specified IDs for them in the Model tab.

Attribute	What to do	Notes
Location	Specify the location of the data node to delete via an XPath expression.	You can type the expression in the choice box, or click the ellipsis to launch the XPath Navigator. Valid expressions include: ◆ last()—specifies the last item in the nodeset. ◆ 1—specifies the first item in the nodeset. ◆ index()—Refers to the position of the current selection in the specified repeat.

Dispatch

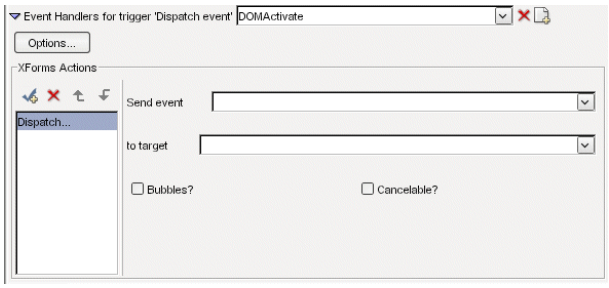
Description

Dispatches XForms events to a named element on a form. You can dispatch events that are:

- Standard XForms events—like xforms-enabled, xforms-disabled, and so on.
- Custom XML events—A custom event that you've defined for the form.

Attributes

To construct the action, use the Event Editor to define the attributes.



Complete the attributes as follows:

Attribute	What to do	Notes
Send event	Choose an event to dispatch.	If the event is not listed, you can type it in the choice box.
target	Choose the ID of the element the event is dispatched to.	The choice box is populated with elements for which you have defined IDs. Valid target elements include: ◆ Model elements ◆ Instance elements ◆ Action elements ◆ Submission elements ◆ Form controls
bubbles	Check if the dispatched event bubbles.	For custom events only. Optional.
cancelable	Check if the dispatched event can be cancelled.	For custom events only. Optional.

Insert

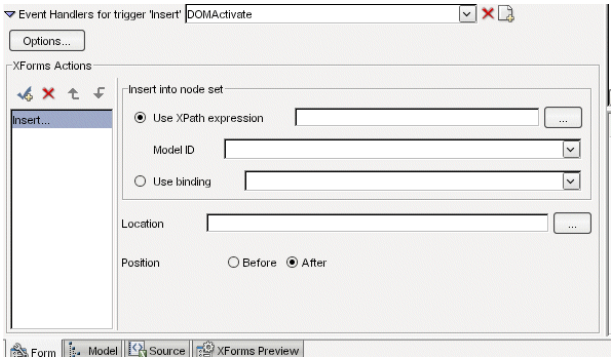
Description

Inserts a new node of instance data. The XForms Specification requires that the instance data be a homogeneous collection (typically, a repeat block).

By default, the inserted node is a duplicate of the last node in the nodeset.

Attributes

To construct the action, use the Event Editor to define it's attributes:



Complete the attributes as follows:

Attribute	What to do	Notes
Use XPath expression	Click **Use XPath expression** to specify an XPath expression that identifies the data node to insert.	You can type the expression in the choice box, or click the ellipsis to launch the XPath Navigator.
Model ID	Optional. Choose a Model ID from the choice box.	A Model ID is only needed when: ◆ You click **Use XPath expression**. ◆ The form references multiple models containing instance data elements of the same name.
Use binding	Click **Use binding** to specify an existing bind expression that identifies the data node to delete, then choose a binding ID from the choice box.	The choice box is only populated if you've defined a binding element **and** specified an ID for it (in the Model tab).
Location	Specify the location for the insert, within the data node, via an XPath expression.	You can type the expression in the choice box, or click the ellipsis to launch the XPath Navigator. Valid expressions include: ◆ last()—specifies the last item in the nodeset. ◆ 1—specifies the first item in the nodeset. ◆ index()—specifies the position of the current selection in the specified repeat.
Position	◆ Choose **Before** to insert the new node before the node specified by the location attribute. ◆ Choose **After** to insert the new node after the node specified by the location attribute.	

Load link

Description

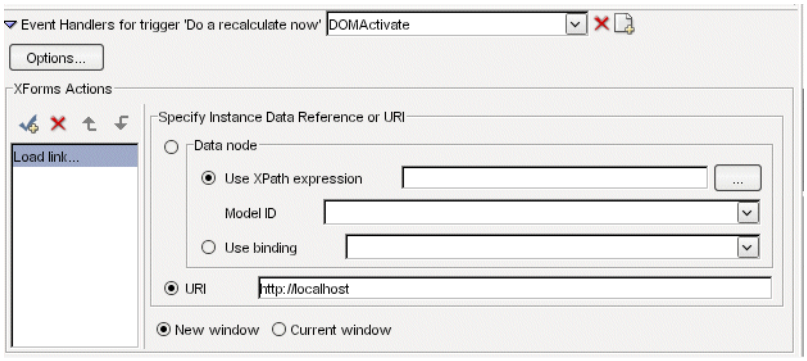
Navigates an URL in the same or different browser window.

To test the Load link action, use View in browser option (available from the XForms Preview tab). The Load Action is not supported in Preview mode.

If the link fails at runtime, no navigation occurs and the xforms-link-error event fires. You can set up an event handler for the xforms-link-error event to intercept and handle this error gracefully.

Attributes

To construct the action, use the Event Editor to define it's attributes:



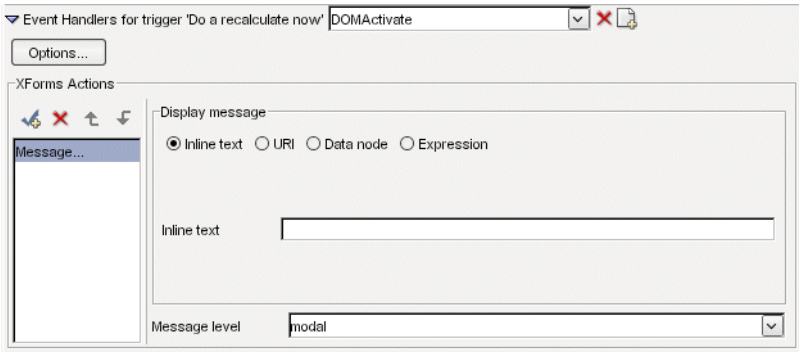
Complete the attributes as follows:

Attribute	What to do	Notes
Data node	Choose to specify a data node that resolves to a URI.	The URI must include the URL scheme such as HTTP or HTTP.
	Choose **Use XPath expression** to specify the URI via an an XPath expression that resolves to a URI.	You can type the expression, or click the ellipsis to launch the XPath Navigator. A Model ID is only needed when: ◆ You click **Use XPath expression**. ◆ The form references multiple models containing instance data elements of the same name.
	Click **Use binding** to specify an existing bind expression that resolves to a URI.	The choice box is only populated if you've defined a binding element **and** specified an ID for it in the Model tab.
URI	Choose when you want to manually specify the URL.	The URI must include the URL scheme such as HTTP or HTTPS. For example: `http://www.novell.com`
New window	Choose if successful navigation should launch the URL in a new window.	
Current window	Choose if successful navigation should launch the URL in the current window.	

Message

Description Launches a modal, modeless, or ephemeral message box.

Attributes To construct the action, use the Event Editor to define it's attributes:



Complete the attributes as follows:

Attribute	What to do	Notes
Display Message	Choose one:	
	Inline text	Use for simple, static messages. Type the message you want displayed in the text box.
	URI	Use to display the contents of the file at the specified URL.The URL is invoked and the raw text is displayed in an alert box. For example: `http://www.myserver.com/error-msg/data-error.txt)` **TIP:** You might use this as an alternate way of storing (and localizing) error messages; the URL could even pass arguments to a CGI script or a servlet to fetch localized messages.
	Data node	Displays a message stored in an instance data node.
	Expression	Use to construct complex inline text messages. Constructs an XForms output element as a child of the xforms:message element. For example: `if(string-length("account-info/account-number")>4),'You have entered too many characters.', 'The data is invalid')`
Message level	Choose one:	
	Modal	Blocks user input to all other windows until the user dismisses the message.
	Modeless	Allows users to work with other windows without having to respond to the message.
	Ephemeral	

Rebuild, Recalculate, Refresh, and Revalidate model

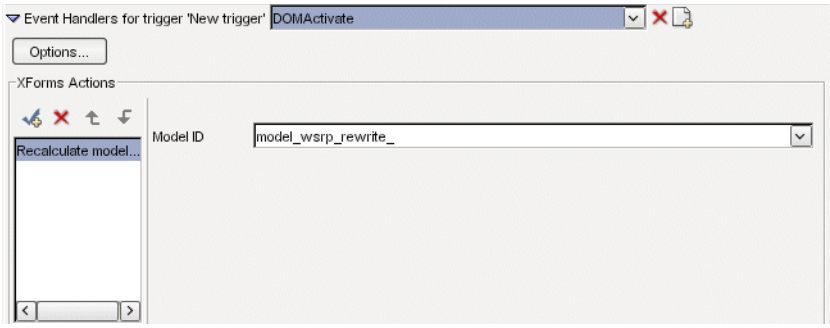
Description

Forces xforms-rebuild, xforms-recalculate, xforms-refesh and and xforms-revalidate events to occur. Use the corresponding actions to specify a behavior different from the default for these events.

📖 For more information on XForms event processing flow, see the xforms-rebuild, xforms-recalculate, xforms-refresh, and xforms-revalidate events in the XForms 1.0 Specification.

Attributes

To construct the action, use the Event Editor to define it's attributes:



Complete the attributes as follows:

Attribute	What to do	Notes
Model ID	Choose the Model ID from the choice box.	This is the model that will receive the event.

Reset model

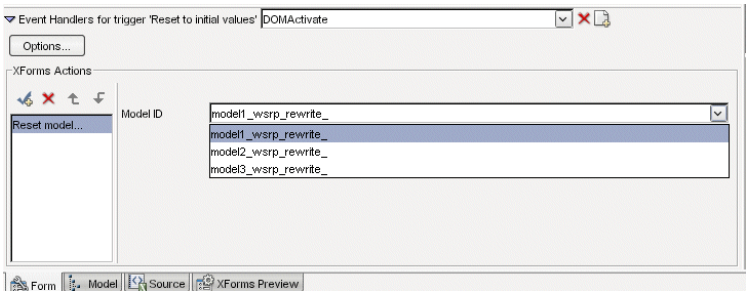
Description

Sets the instance data, of the specified model, to the values at form initialization.

📖 For more information on XForms event processing flow, see the xforms-reset event in the XForms 1.0 Specification.

Attributes

To construct the action, use the Event Editor to define it's attributes:



Complete the attributes as follows:

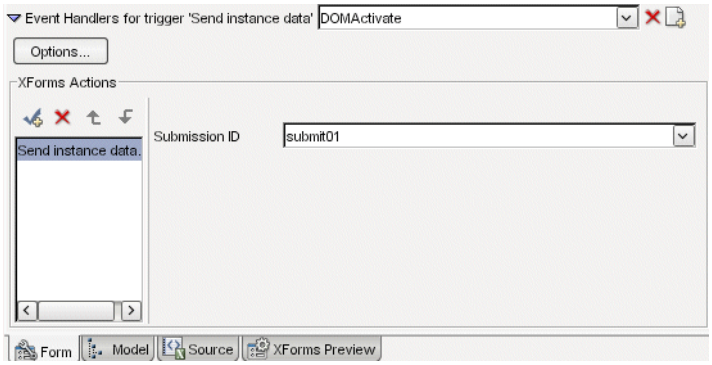
Attribute	What to do
Model ID	Choose the ID of the Model you want reset

Send instance data

Description

Forces the form to begin submit processing.

Attributes To construct the action, use the Event Editor to define it's attributes:



Complete the attributes as follows:

Attribute	What to do	Notes
Submission ID	Choose a submission ID	Submission elements are defined in the Model tab. The choice box is only populated when a Submission element is created and given an ID.

Set Focus

Description Use Set Focus to move focus to a specific form control.

Attributes To construct the action, use the Event Editor to define it's attributes:

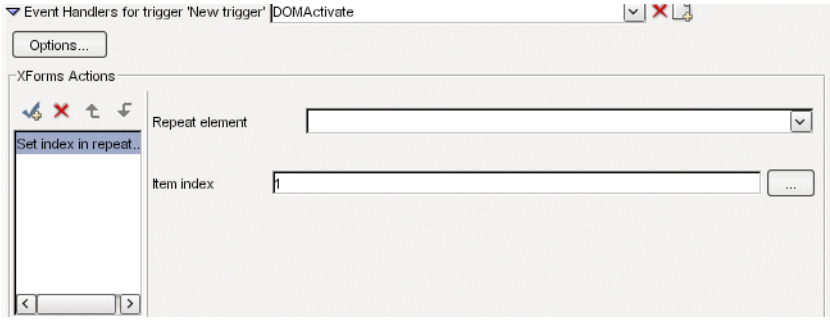
Complete the attributes as follows:

Attribute	What to do	Notes
Control ID	Choose a Control ID from the choice box.	You can define a Control ID in the Property Inspector (in the Form tab). The choice box is only populated when control elements are named in the Property Inspector.

Set index in repeat

Description Use the Set index to specify the current node of a repeat block.

Attributes To construct the action, use the Event Editor to define it's attributes:



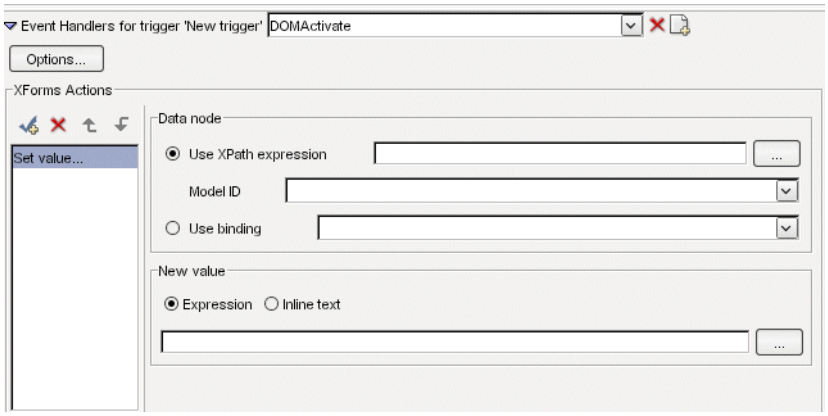
Complete the attributes as follows:

Attribute	What to do
Repeat element	Choose a repeat element from the choice box.
Item index	Specify an XPath expression that defines index

Set value

Description Use to set the value of an instance node.

Attributes To construct the action, use the Event Editor to define it's attributes:



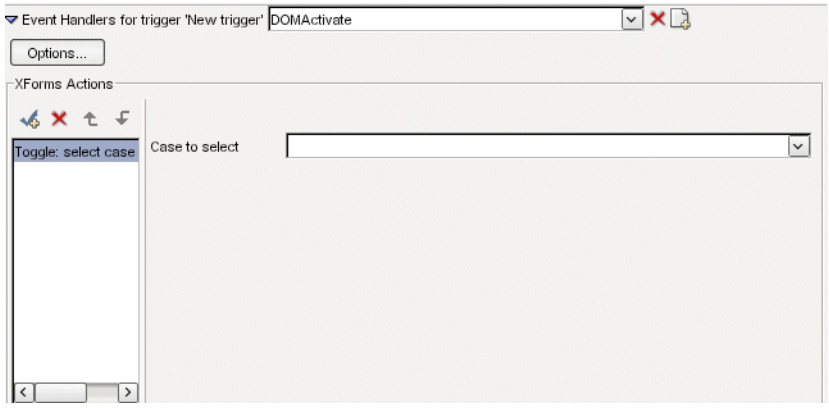
Complete the attributes as follows:

Attribute	What to do	Notes
Data node	Choose one to specify the data node whose value you want to set.	
	Use XPath expression—to specify an XPath expression.	You can type the expression in the choice box, or click the ellipsis to launch the XPath Navigator. A Model ID is only needed when the form references multiple models containing instance data elements of the same name.
	Use binding—to specify an existing bind expression.	The choice box is only populated if you've defined a binding element **and** specified an ID for it in the Model tab.
New Value	Choose one:	
	Inline text—to specify static values.	For example: `This is a test value.`
	Expression—to construct dynamic values based on user input or other elements not known at design time.	For example: `concat('Test: ', /data:test/data:form_data/data:instance_node)`

Toggle select case in switch

Description Use to specify the case to display in a Switch element.

Attributes To construct the action, use the Event Editor to define it's attributes:



Complete the attributes as follows:

Attributes	What to do	Notes
Case to select	Select the case ID from the choice box.	The choice box is only populated if you've defined a Switch element.

Customizing event handlers

➢ **To customize event dispatching:**

1 Access the Advanced Options dialog by:

- Clicking **Advanced options** when you first choose the event you want to dispatch

  OR

- Clicking **Options** from the Event Editor.

2 Complete the dialog as follows:

Option	Description
Handle event at observer element	Choose an observer element from the list box. This option moves the event handler to the selected XML element. **NOTE:** You should not put an event observer on the body tag. To achieve the same result, click **Attach event handler to model**, select a model, and specify the body element as the observer.
Handle event during phase	Lets you control the order in which event handlers are executed.
Restrict handler to events on selected element	This is valid only when the event handler is attached to an ancestor of the selected control; it makes the handler specific to the selected control. You can use this option to cause something to happen at a particular point in the event-handling sequence for the specified control.
Continue event propagation after handlers at this element	When used in combination with handlers on ancestor nodes and phase options, this is a way to control which event handlers are executed for a particular event.
Perform or cancel default action button	Provides a means of overriding the default behavior of the XForms processor. Choosing cancel always cancels the default behavior—no conditional processing is allowed. (Not all XForms events allow you to cancel the default behavior.)

3 Click **OK**.

Testing forms

You'll want to test your form's data processing logic and its look and feel. The Form Designer provides built-in tools for each of these tasks:

Use this tool	To test this
XForms preview	Event and data processing XHMTL component look and feel
View Form in Browser	The look and feel of all components on the page

Using XForms Preview

XForms Preview uses a Swing renderer to render the XForms controls on the page. This feature is especially useful for testing the structure of your submitted data.

➢ **To use XForms Preview:**

1 Open the form.

2 Choose the **XForms Preview** tab.

The form is rendered and is available for interaction:

- Any controls that are unexpectedly disabled (you didn't mark them disabled in your code) might indicate a data binding problem.
- You can view submission nodes in the left pane of the XForms Preview.

XForms Preview limitations The XForms Preview does not display any HTML associated with the XHTML page.

Using View Form in browser

The View Form in browser uses an XHTML renderer to display both the XHTML and the HTML on a single page. This feature is especially useful to help you work out any layout issues.

To use View Form in browser, your form's project must be deployed to an application server. Once your project is deployed, changes that you make to the form are immediately available—you don't have to redeploy your project.

➢ **To use View Form in browser:**

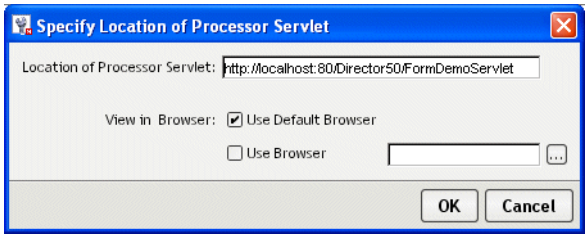
1 Open the project.

2 Open the form you want to view.

3 Choose **XForms Preview**.

4 Choose the View Form in browser button.

The Specify Location of Processor Servlet dialog displays:



5 Complete the dialog using the following values:

Field	What to specify
Location of processor servlet	Specify the name of the server where the Form's project is deployed.
Use default browser	Choose this radio button if you want the Form Designer to locate the default browser for the current machine.
User browser	Choose this radio button and the corresponding text field to specify a browser other than the default.

6 Click **OK**.

The form displays in the browser. If you receive a Page Not Found error, make sure that the form's project has been deployed to the server specified.

📖 For more information on deploying, see the chapter on deploying in *Developing exteNd Director Applications*.

Testing portal themes

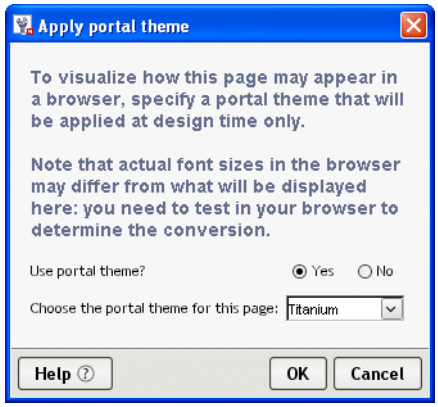
You may want to test what your form would look like using various portal themes.

➢ **To test what your form would look like using a portal theme:**

1 Open the form in the **Form** tab.

2 Choose **Form Designer>Set Portal CSS Theme**.

The Apply Portal Themes dialog displays:



3 Complete the panel as follows:

Field	What to specify
Use portal theme?	Choose yes to apply a theme.
Choose the portal theme for this page	Choose the portal theme to apply.

4 Click **OK**.

This styling applies only at design time. The Form Designer deletes the link to the theme file when the file is saved and restores it when the file is reopened (in the Form Designer). To remove the link completely, reopen the dialog and choose No.

The portal theme is not used by XForms Preview or View form in browser.

Testing browser font sizes

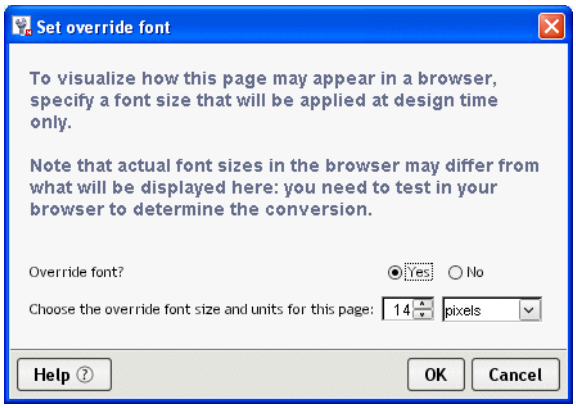
You may want to test your form with various font sizes.

➢ **To test browser font sizes at design time:**

1 Open the form in the **Form tab**.

2 Choose **Form Designer>Set browser font override**.

The Set override font displays:



3 Complete the dialog as follows:

Field	What to specify
Override font?	Choose yes to override the browser's font size in the design environment.
Choose the override font size and units for this page	Specify the font size and units to be used.

4 Choose **OK**.

The font you specify is applied as a font size style on the <body> tag.

This styling applies only at design time. The Form Designer deletes the link to the theme file when the file is saved and restores it when the file is reopened (in the Form Designer). To remove the link completely, reopen the dialog and choose No.

Validating the form's XML structure

You can validate the form to ensure that its XML is well formed and that it conforms to the schemas for the declared namespaces that can be resolved.

➢ **To validate a form:**

1 Select **Form Designer>Validate**.

7 Database Pageflow Wizard

This chapter provides instructions for using the Database Pageflow Wizard. It contains the following sections:

About the Database Pageflow Wizard

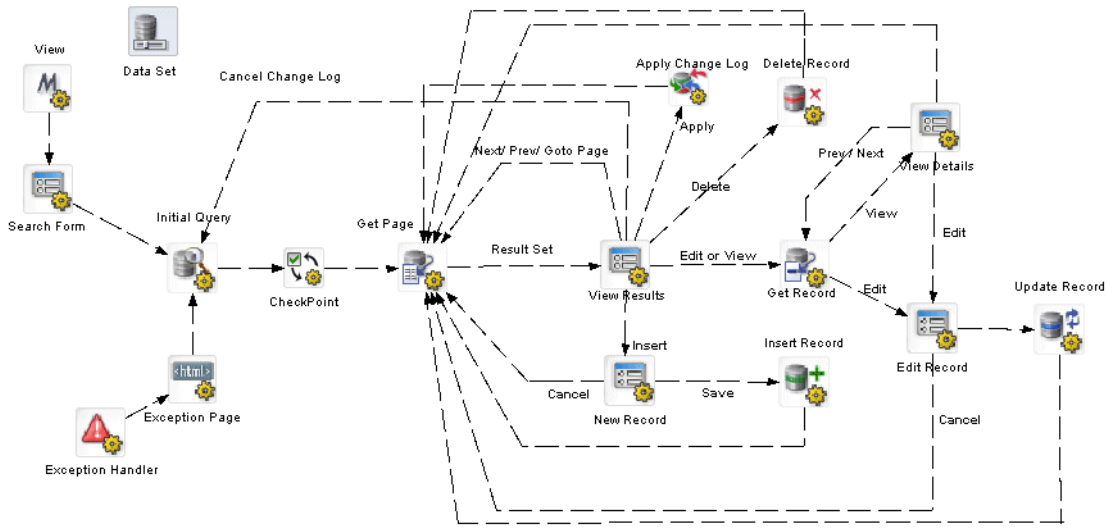
exteNd Director provides the **Database Pageflow Wizard** to help you create database pageflows. A *database pageflow* is a flow that gives the user a way to find, display, and modify records in a database.

The Database Pageflow Wizard lets you create flows that access a single database table, or flows that navigate to other tables that are related by many-to-one relationships (lookups) and/or one-to-many relationships (master/detail).

After you use the wizard, you can run the generated pageflow right away. You can also use the Pageflow Modeler to make changes to the pageflow, just as you would with any other pageflow.

The Database Pageflow Wizard generates a set of forms (XHTML pages that use XForms technology) as well as one or more pageflow processes that tie the forms together into a simple database application. This application provides a convenient and easy-to-use interface for accessing one or more database tables.

Flows that access a single table The following example shows a typical database pageflow used to access a single table:

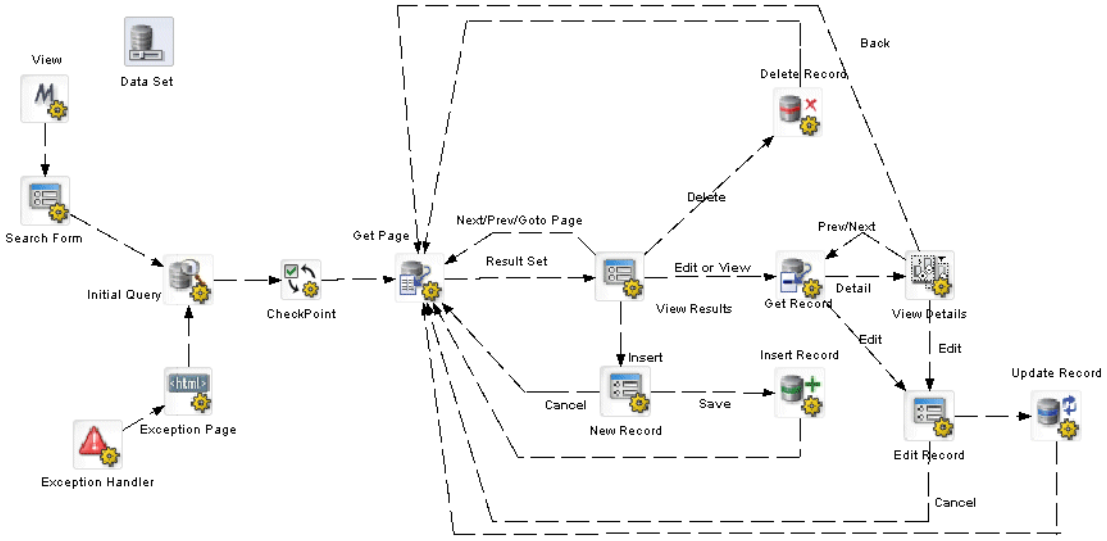


Master/detail database flows When you build a database pageflow, you specify which table is the **primary table**. When the wizard knows which table is primary, it can examine the relationships among the various tables you've selected to determine how many levels there are within the query. A lookup (many-to-one relationship) is treated as a single level within the query, whereas a master/detail (one-to-many relationship) is treated as two levels.

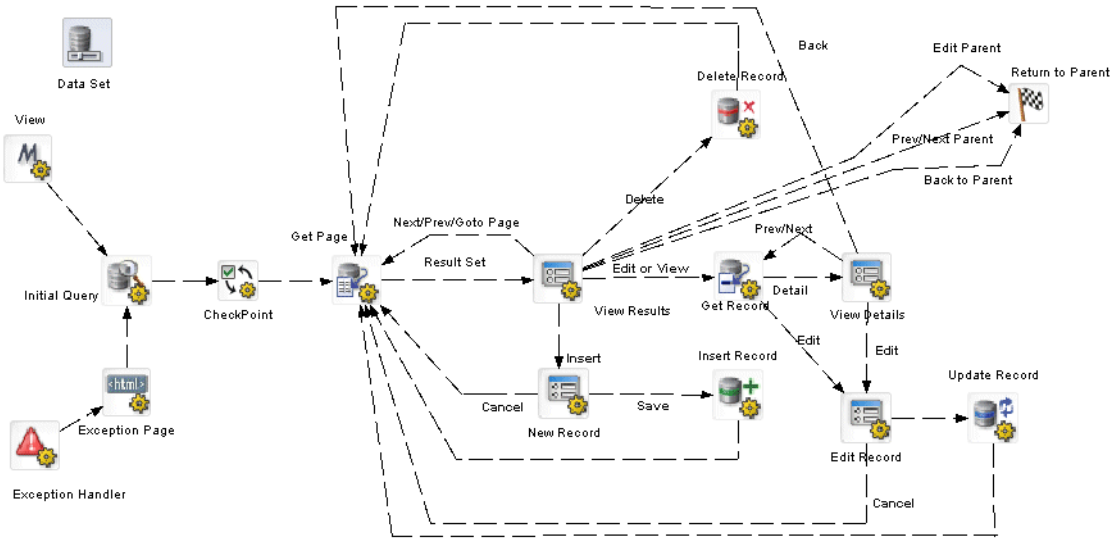
When you build a database pageflow that combines data from tables that have a master/detail relationship, the wizard generates **multiple pageflow processes**, one for each level within the query.

The flow that accesses data in the primary table is the parent flow. The parent flow has a Pageflow activity (View Details) that references a child flow. The child flow accesses data in the detail table. When the user clicks the View link, the parent flow reaches its Pageflow activity, which passes control to the child flow. When the user clicks on the Back link, the child flow reaches its Finish activity, which passes control back to the parent flow.

Here is an example of a parent flow:

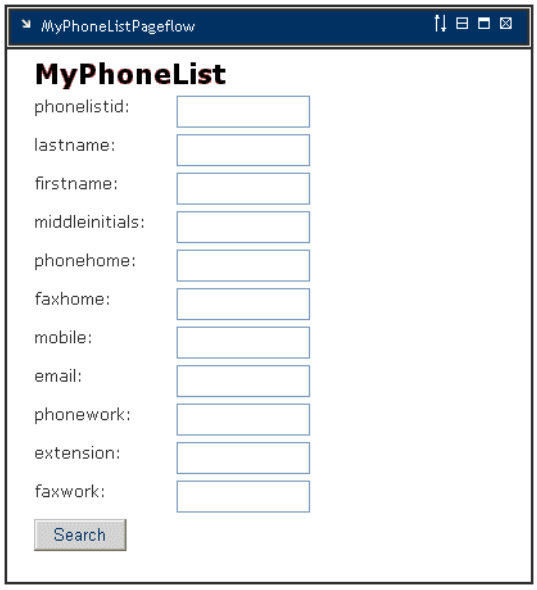
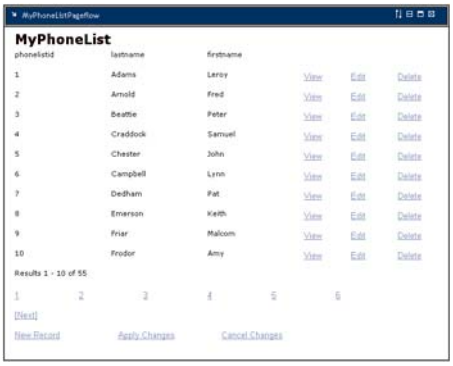


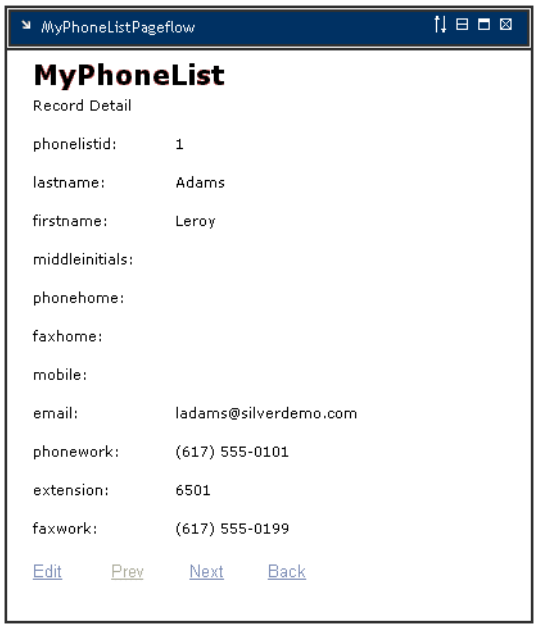
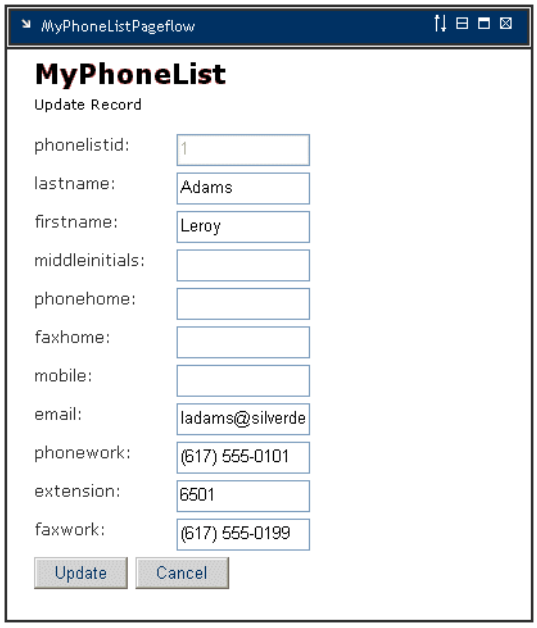
Here is an example of a child flow that is referenced by the View Details activity:

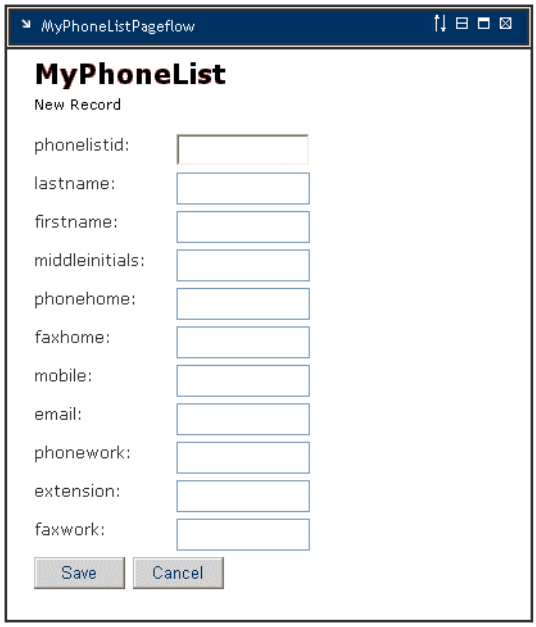


Forms

When you create a database pageflow, the following forms (XHTML files) are generated for you:

Form	Description
Search Form	Provides a user interface for entering search criteria. When the user presses the Search button, the search terms are sent to the server as an XML document. 
View Results	Displays summary data for a single page of search results. The summary data are presented in tabular format. This form has View, Edit, and Delete buttons that let the user perform operations on the currently selected row. It also has a New Record button that lets the user insert a new database record. When you indicate that you want to use a change log to store modifications, the View Results form also includes Apply Changes and Cancel Changes buttons. 

Form	Description
View Details	Displays details for one record in the result set. It also contains an Edit button to allow the user to modify the values associated with the current record.  In a master/detail situation, the View Details form also displays the results view of the child table in a tabular format. It shows all of the records of the child table that are linked to the current parent record.
Edit Record	Provides a user interface for editing the detail fields associated with the primary table. The Edit Record form also contains Update and Cancel buttons. The Edit Record form automatically sets a validation on each field to ensure that the user does not try to enter a string that is longer than the width of the associated database column.  **NOTE:** The Edit Record form does not allow you to modify columns in related tables.

Form	Description
New Record	Provides a user interface for creating a new record in the primary table. This interface provides fields for entering the column values associated with the new record. This form is very similar to the Edit Record form. The main difference is that the New Record form lets the user specify values for key fields (except for those that are not automatically incremented.) This form includes a Save button to allow the user to add the new record to the database.  The New Record form does not allow you to enter values for columns in related tables.

Pageflows

Every pageflow process generated by the Database Pageflow Wizard includes a Data Set object that provides all of the information required to access the database.

For more information on the Data Set, see "Working with the Data Set" on page 148.

Each pageflow process created by the Database Pageflow Wizard includes a Form activity for each form created. In addition, the flow contains the following database activities:

Activity	Description
Initial Query activity	Obtains the keys of all records from the database that match the search criteria that the user specified on the Search Form. This activity takes the XML document that contains the search terms (which is generated by the Search Form) as input. Once the keys have been retrieved from the database, this activity stores the result set in a record cache.

Activity	Description
Get Page activity	Retrieves the summary data for a single page of records. The Database Pageflow Wizard lets you control which columns are included in the summary data and specify how many records are included in each page. The Get Page activity uses the set of keys returned by the Initial Query activity to retrieve the records for one page of results. The order of records in the summary data is based on the order established when the Initial Query activity was executed, not by the order in which records are retrieved by the Get Page activity.
Get Record activity	Retrieves the detail fields for a single record.
Record Insert activity	Sends the data for the new record to the database by executing a SQL INSERT statement. **NOTE:** If you enable the change log, the SQL INSERT is not actually performed until the Apply Change Log activity is executed. Here's an example of the kind of statement that might be executed: `INSERT INTO employees (firstname, lastname, city, state, phone) VALUES ('Bob', 'Jones', 'Boston', 'MA', '617-555-9999')` If the insert succeeds, the flow continues on to the next activity. Otherwise, an error message is displayed.
Record Update activity	Sends the updated data for one record to the database by executing a SQL UPDATE statement. To support optimistic concurrency control, the UPDATE statement includes the original (cached) values in the WHERE clause. If the WHERE clause fails to match any rows, that means that the data has been changed by another user, so a data concurrency exception is thrown to indicate this to the user. **NOTE:** If you enable the change log, the SQL UPDATE is not actually performed until the Apply Change Log activity is executed. Here's an example of the kind of statement that might be executed: `UPDATE employees SET city = 'Waltham', phone = '781-484-8200' WHERE firstname = 'Joseph' AND lastname = 'Smithe' AND city = 'Boston' AND state = 'MA' AND phone = '617-555-1213' AND employeeid = 61` If the update succeeds, the flow continues on to the next activity. Otherwise, an error message is displayed.
Record Delete activity	Deletes the records selected by the user by executing a SQL DELETE statement. To support optimistic concurrency control, the DELETE statement includes the original (cached) values in the WHERE clause. If the WHERE clause fails to match any rows, that means that the data has been changed by another user, so a data concurrency exception is thrown to indicate this to the user. **NOTE:** If you enable the change log, the SQL DELETE is not actually performed until the Apply Change Log activity is executed. Here's an example of the kind of statement that might be executed: `DELETE FROM employees WHERE firstname = 'Joseph' AND lastname = 'Smithe' AND city = 'Boston' AND state = 'MA' AND phone = '617-555-1213' AND employeeid = 61` If the deletes succeed, the flow continues on to the next activity. Otherwise, an error message is displayed.

If you specify that you want to use a change log, the Apply Change Log activity is also included in the flow:

Activity	Description
Apply Change Log activity	Applies all changes from the record cache to the database. For more information on the Apply Change Log activity, see "Apply Change Log activity" on page 42.

The pageflow also contains the following additional activities:

Activity	Description
View	Mode activity that places the pageflow in View mode.
CheckPoint	CheckPoint activity that handles page refreshes within the flow. The CheckPoint activity acts as a transaction marker, indicating the starting point for processing whenever the user refreshes the View Results, View Details, Edit Record, and New Record forms.
Exception handler	Exception activity that handles all exceptions thrown during the course of processing.
Exception page	An HTML activity that displays exception information in an HTML page.

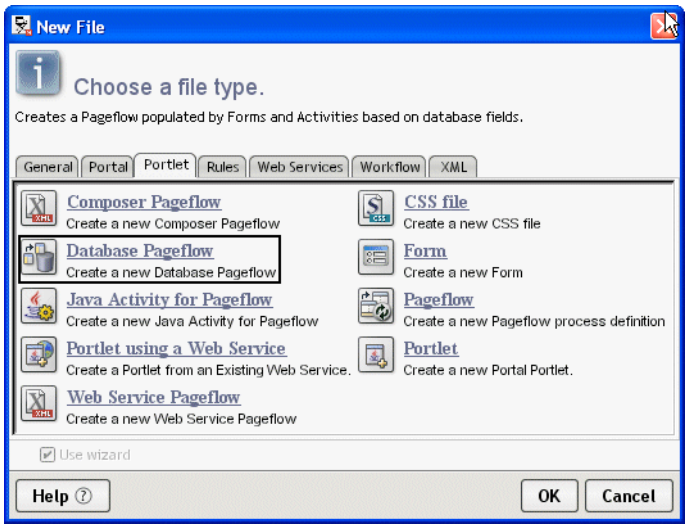
Using the Database Pageflow Wizard

➢ **To create a database pageflow process:**

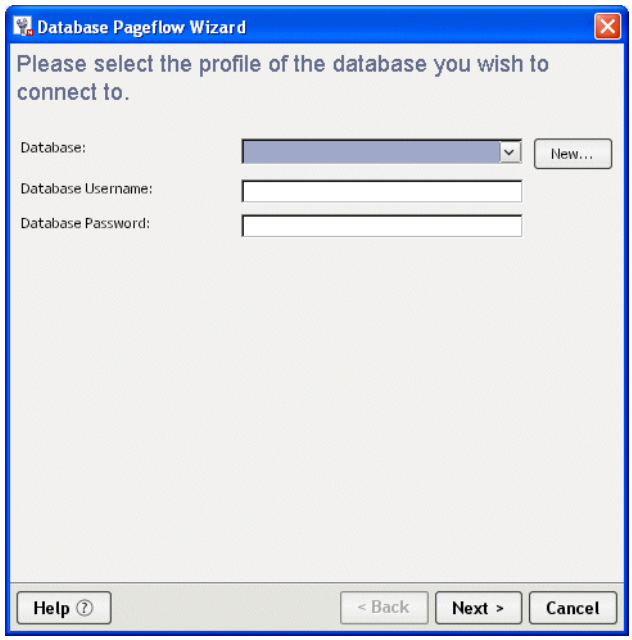
1 With your project open in exteNd Director, select **File>New**.

2 Click the **Portlet** tab.

3 Select **Database Pageflow** and click **OK**:



4 If you do not have a profile for the database you want to connect to, create one by clicking the **New** button.

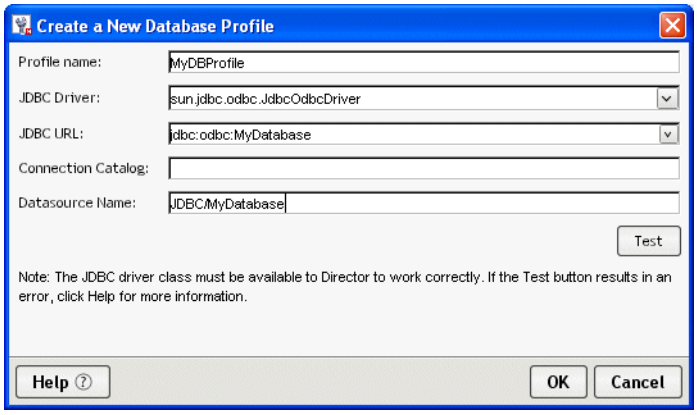


4a Specify settings in the Create a New Database Profile dialog as follows:

Setting	Description
Profile name	Enter any name to identify the profile.
JDBC Driver	Enter the class name of the JDBC driver. You can specify any JDBC 2.0-compliant driver. To use the Sun JDBC-ODBC bridge driver (which is included in the JRE), specify **sun.jdbc.odbc.JdbcOdbcDriver**. If you specify a JDBC driver other than Sun's bridge driver, make sure the driver class can be loaded by the development environment. **NOTE:** The jConnect driver should be used with Adaptive Server Anywhere instead of the Sun ODBC/JDBC bridge driver. To use MySQL, specify **com.mysql.jdbc.Driver**.
JDBC URL	Enter an URL that specifies the database you want. For example, you might specify **jdbc:odbc:TestDB** (if TestDB were your ODBC data source name). For a MySQL database, you might specify **jdbc:mysql://localhost:63306/ExpressPortal?user=root&password=novell** The text you enter after the first colon is driver specific.
Connection Catalog	(Optional) Specify which SQL catalog (subset) of the database to connect to—for example, **PayrollDb**. If your database driver does not support catalogs, it will ignore this request. If supported, the connection catalog lets you set up which database tables are retrieved. Connection catalogs are useful when you are connecting to a very large database or only want to connect to a subset of database tables (for example, to exclude production database access).

Setting	Description
Datasource Name	Specify the name of the data source to associate with this database profile. You can specify either the datasource or the full JNDI specification. For example, you might specify **JDBC/ExpressPortal** to use a connection pool called ExpressPortal.

A filled-in panel might look something like this:



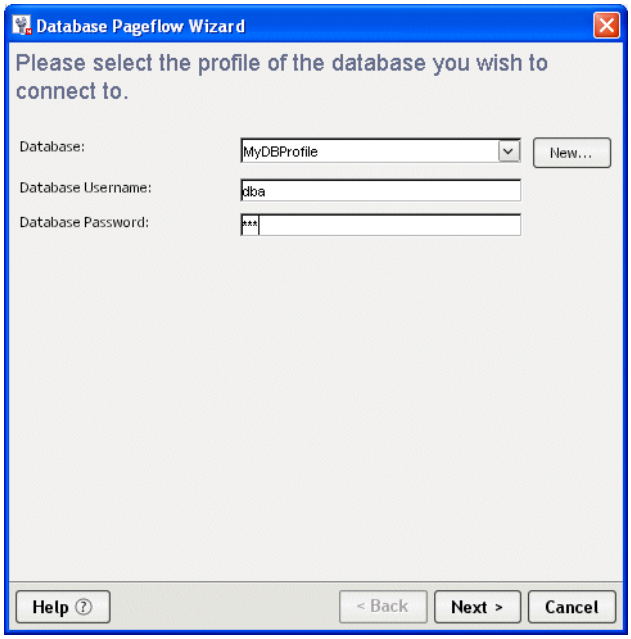
4b Click **Test** to check the connection to the database specified by the JDBC URL.

This test makes a JDBC connection to the database. The test will fail when a connection is not available or a setting is not correctly specified.

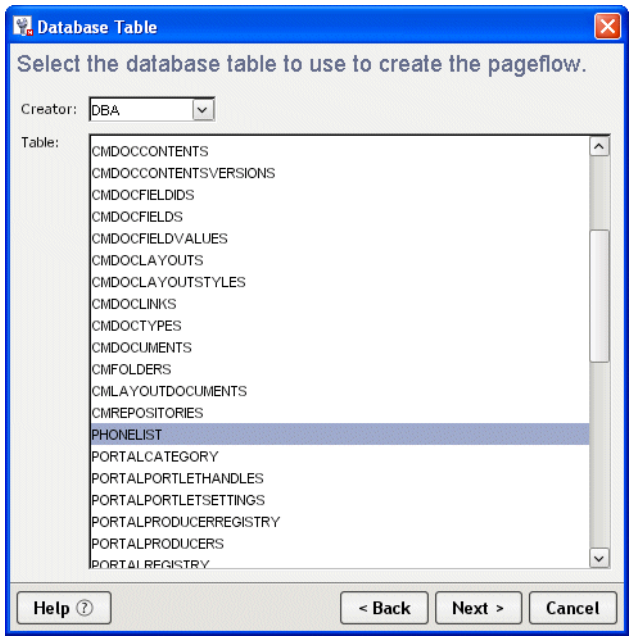
4c On the test popup, enter your database user name and password and click **OK** to verify access.

4d Click **OK** to close the Create a New Database Profile dialog.

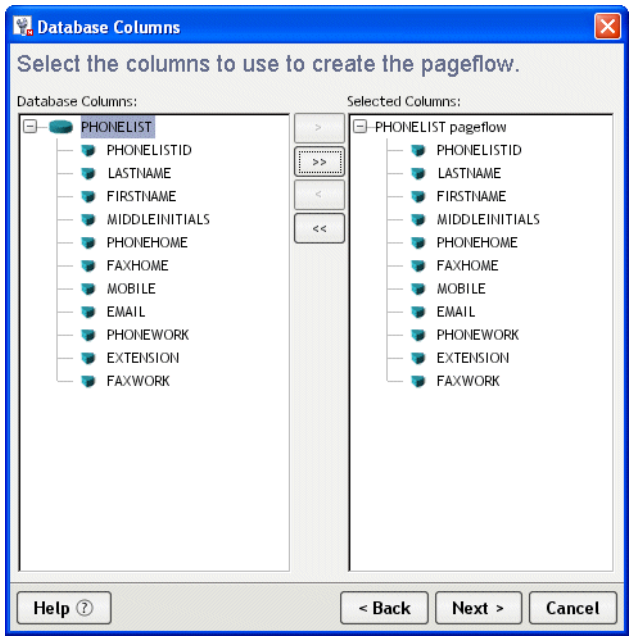
5 Optionally enter your database user name and password on the Database Profile dialog and click **OK**:



6 Select the primary table for your pageflow and click **Next**:



7 Select the columns you want to use in your database pageflow in the **Database Columns** dialog and click **OK**:



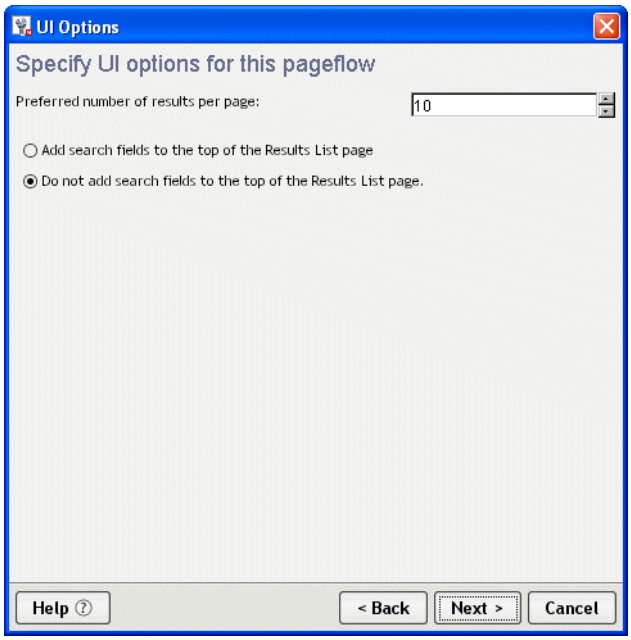
Any tables directly related to the primary table are listed as choices in the Database Columns box, along with columns defined on the primary table.

To select columns for a related table, first open the table in the Database Columns box, then select the columns.

8 Specify the user interface options for the flow as follows:

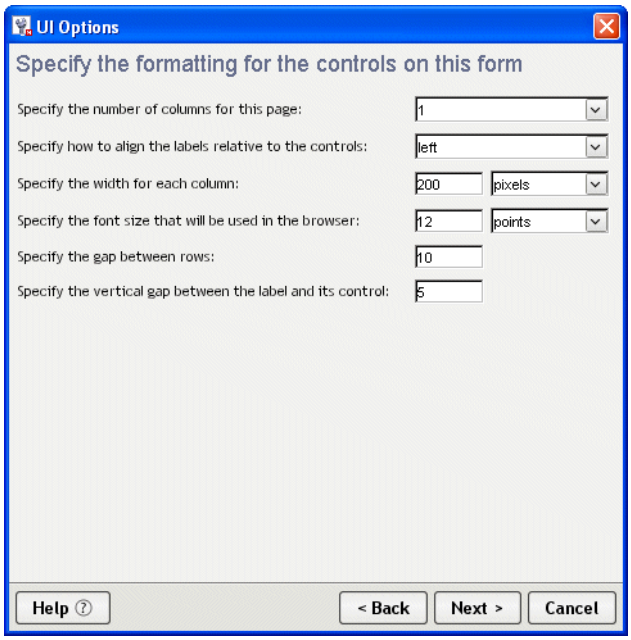
8a Specify how many rows you want to show on each Results List page in the **Preferred number of results per page** field.

8b Indicate whether you want the wizard to add search fields to the top of the Results List page.



8c Click **Next**.

9 Specify how you want to format the controls on forms:



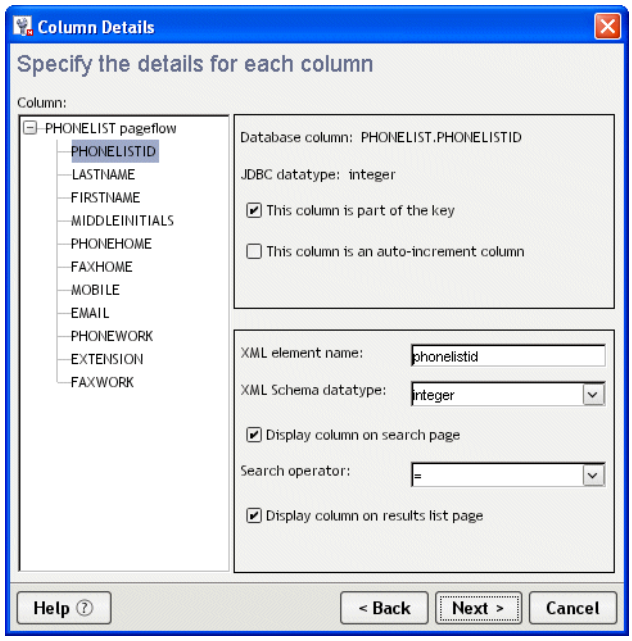
These settings control the appearance of controls on the Search, View Details, Edit Record, and New Record forms.

10 Click **Next**.

11 Select the details for each column as follows:

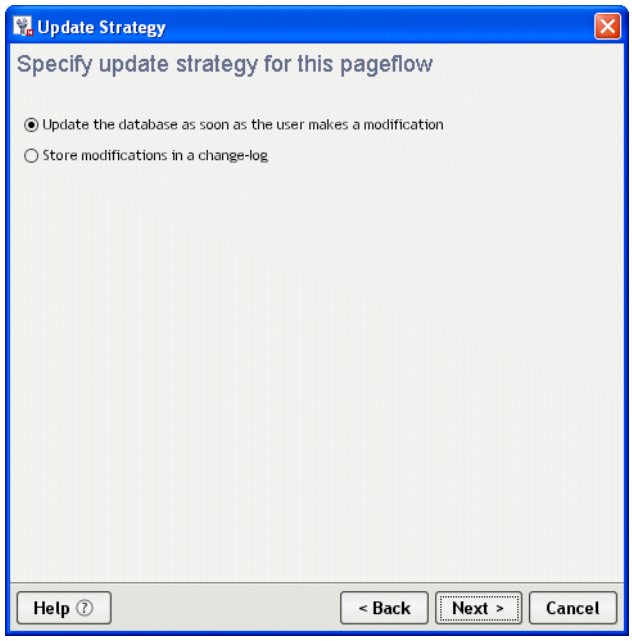
11a Select a column in the Column list.

11b Specify the details for the column:



12 Click **Next**.

13 Specify the Update Strategy and click **Next**:



14 Specify the base names for the files that will be generated. You need to specify a separate base name for each database table you selected:

15 Click **Finish**.

Modifying a database pageflow

After you run the wizard, you use the Pageflow Modeler to make changes to the pageflow. You can also use the Form Designer to make changes to the generated forms.

General guidelines for editing a database pageflow

Here are some things to keep in mind when editing a generated database pageflow:

- **When you make a change to a pageflow**, this does not change any of the forms used by the flow.
- **If you want to add or remove a column in a database pageflow**, you need to manually add or remove the corresponding field in one or more forms. To add or remove a database column, you need to make changes in the property sheet for the Data Set. Once you've done this, you typically need to add the column to the design-time instance data in each form, as well as add the input or output control to the form.

  For complete details on making changes in the property sheet for the Data Set, see "Working with the Data Set" on page 148. For complete details on modifying the instance data or the controls associated with a form, see Chapter 6, "Form Designer".
- **If you prefer not to make manual changes to a database flow**, you can run the Database Pageflow Wizard again. When you run the wizard, you can either overwrite your old flow and forms, or create new ones.

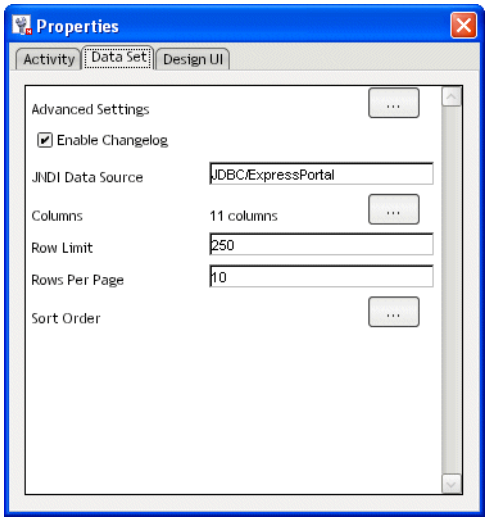
Working with the Data Set

The Data Set provides all of the information that a pageflow requires to connect to a database and access rows and columns of data. The Data Set lets you make changes to various database settings. In many cases, it simply allows you to modify settings you specified when you first ran the Database Pageflow Wizard. In other cases, it lets you make changes to more advanced settings that are not available in the wizard.

Each Data Set has a unique name that is referenced by the following database activities:

- Initial Query activity
- Get Page activity
- Get Record activity
- Record Insert activity
- Record Update activity
- Record Delete activity

Since the Data Set encapsulates all of the information needed to access the database, these activities do not need to specify this kind of information. Instead, they simply point to a Data Set object:



The properties of the Data Set are:

Property Inspector tab	Property name	Description
Activity	Name	Specifies a name for the activity.
	Description	Describes what the activity does.

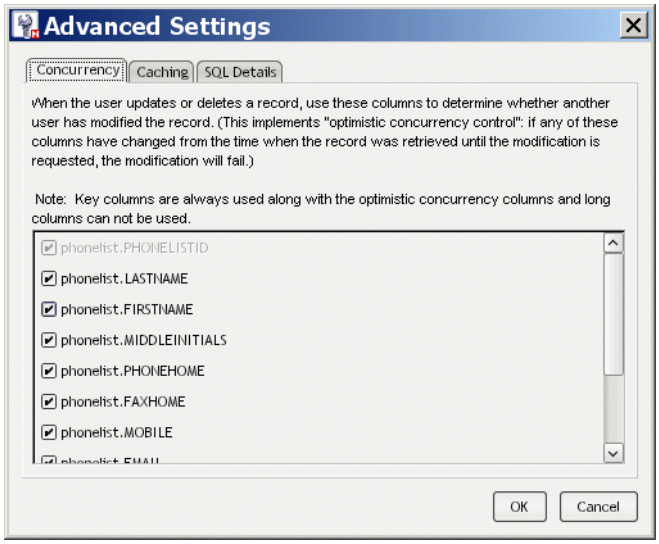
Property Inspector tab	Property name	Description
Data Set	Advanced Settings	Allows you to specify concurrency and caching settings, as well as other details that control database access. 📖 For more information, see “Advanced settings” on page 149.
	Enable Changelog	Indicates whether the change log will be enabled for this flow. Changing this setting does not add or remove activities or links within the flow, nor does it alter the forms associated with the flow. Therefore, if you want to add or remove the Apply Change Logs activity or the Apply Changes links in the forms, you need to make these changes by hand.
	JNDI Data Source	Identifies the data source. The valid data source types are: ◆ `JDBC/xxx` Use this syntax to reference a database that is set up as a connection pool. ◆ `jdbc/ref name` Use this syntax to reference a resource property in your deployment plan. If you do this, you must also have a resource data source in the web.xml file. ◆ `Databases/xxx/DataSource` Use this syntax to reference a database that was added to the exteNd™ Application Server.
	Columns	Lets you add or remove columns from the pageflow or change their properties.
	Row Limit	Specifies the maximum number of rows that will be retrieved.
	Rows Per Page	Specifies the number of rows that will be displayed on each page of the View Results form.
	Sort Order	Allows you to specify the order in which rows will be sorted in the result set. For more information, see “Sort order” on page 151.

Advanced settings

The Advanced Settings dialog lets you specify several settings that are not available in the Database Pageflow Wizard.

Concurrency control

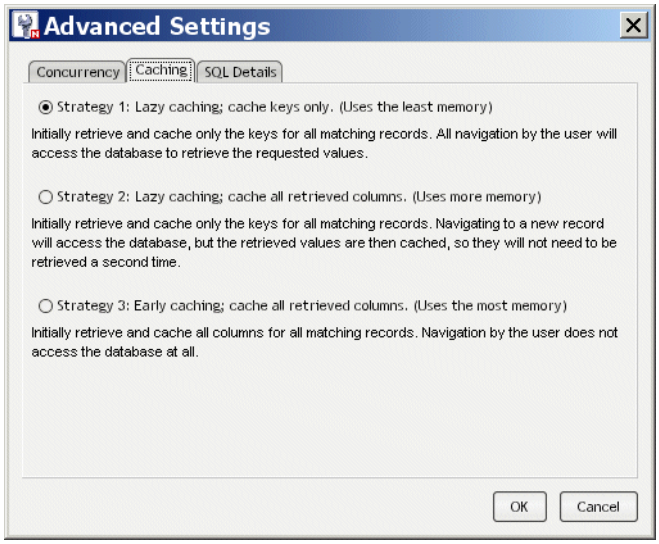
On the Concurrency tab, you can select columns for concurrency checking:



The columns you select are used to verify that no changes were made to the database record since the time it was first retrieved. If any changes were made, an attempt to modify the record will fail.

Caching

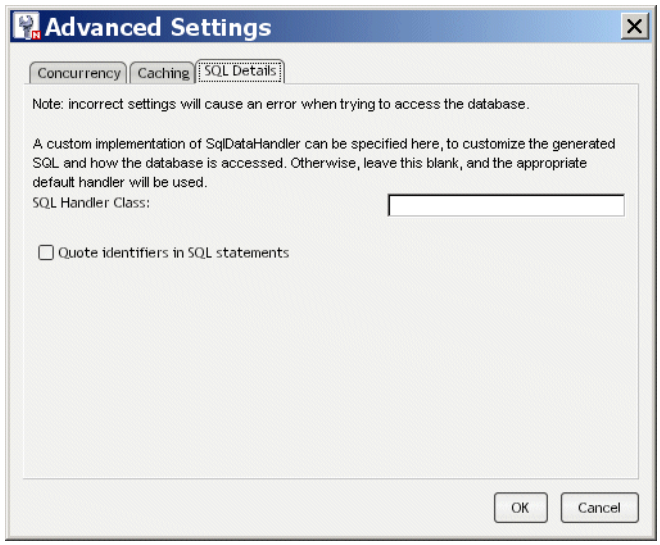
On the Caching tab, you can specify a caching strategy for the flow, as shown below:



The caching strategy used for a database pageflow has a direct effect on how well the flow performs at runtime. The strategy you select will depend on your application requirements. If you're most concerned about memory usage, select Strategy 1. If you're most concerned about minimizing database access operations, select Strategy 3. For a more balanced approach, select Strategy 2.

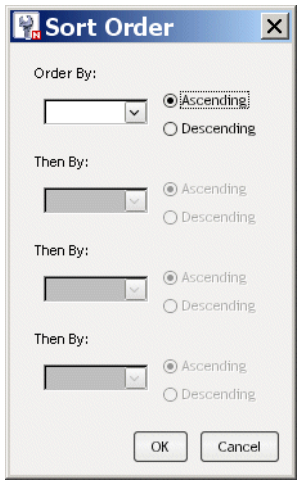
SQL Details

The SQL Details tab provides some additional settings for those who want greater control over how the pageflow accesses the database at runtime. It lets you override the default SQL handler class and also specify whether identifiers will be wrapped in quotes in SQL statements.



Sort order

The Sort Order dialog lets you specify the order in which rows will be sorted in the result set. A database pageflow can have up to four levels of sorting:



8 Web Service Pageflow Wizard

This chapter provides instructions for using the Web Service Pageflow Wizard. It contains the following sections:

About the Web Service Pageflow Wizard

exteNd Director provides the **Web Service Pageflow Wizard** to help you create pageflows that execute Web Services.

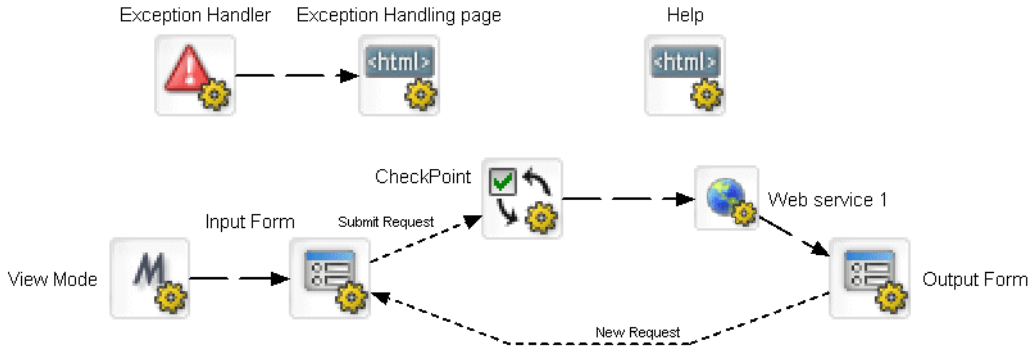
After you use the wizard, you can run the generated pageflow right away. You can also use the Pageflow Modeler to make changes to the pageflow, just as you would with any other pageflow.

The Web Service Pageflow Wizard takes a Web Service Description Language (WSDL) file as input. With the information provided in the WSDL file, it generates a set of forms (XHTML pages that use XForms technology) as well as a pageflow process that ties the Web Service and the forms together into a simple application. This application provides a convenient and easy-to-use interface for invoking Web Service.

NOTE: The Web Service Pageflow Wizard provides support for document-style WSDL files that contain a schema. However, you can create a pageflow that uses an RPC-style Web Service by using a Java activity.

For details on how to use an RPC-style Web Service in a pageflow, see Chapter 11, "Working with RPC-Style Web Services".

Example The following example shows a typical Web Service pageflow for a service that takes an argument:

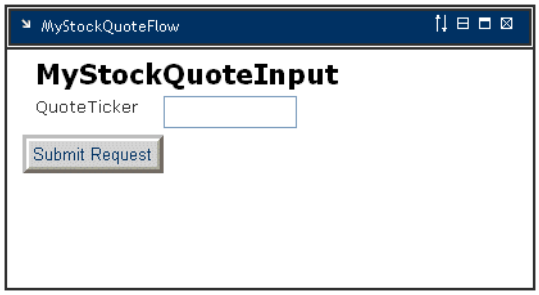


It includes an input form that allows the user to pass a parameter to the service. When the user submits the form, the service executes. Once the service has finished processing, the pageflow displays an output form that shows the data returned by the service.

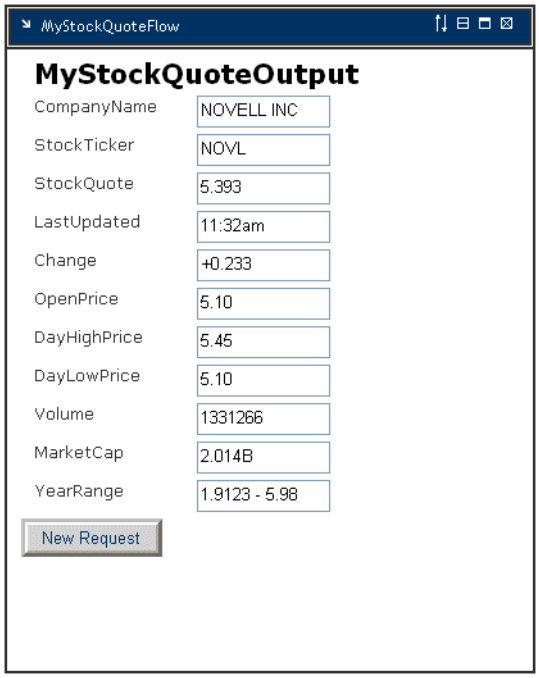
The generated pageflow also includes a CheckPoint activity, which is placed just before the Web Service in the flow. When the page is refreshed (by the user clicking the Refresh button or taking an action on another portlet running on the same portal page), the engine tells the flow to go back to the CheckPoint activity and then execute the Web Service again.

NOTE: To enhance performance in a page that includes several portlets, you may want to remove the CheckPoint activity so that the Web Service is not executed whenever the user refreshes the page or takes an action on another portlet within the page.

What the user sees at runtime At runtime this pageflow would display an input form that allows the user to pass a parameter to the service:



When the user submits the form, the service executes. Once the service has finished processing, the pageflow displays an output form that shows the data returned by the service:



NOTE: If you create a pageflow that executes a Web Service that does not take a parameter, the flow does not include the input form. At runtime the service executes and immediately displays the results of the operation in the output form.

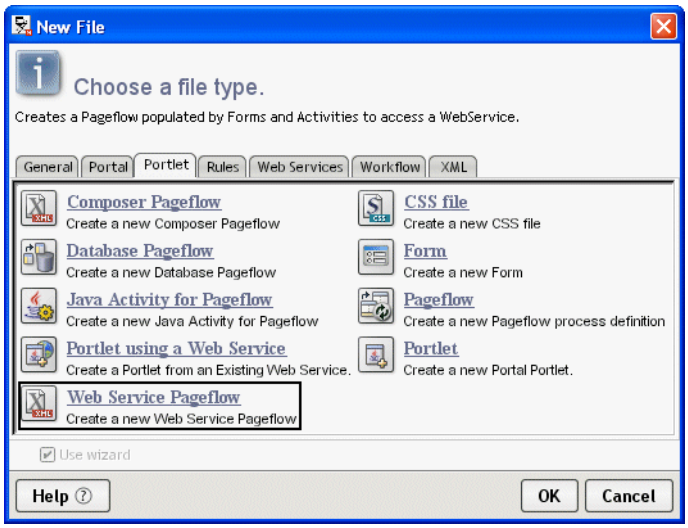
A typical Web Service pageflow also includes a common help page, as well as an exception handler.

📖 For background information on Web Services, see the chapter on Web Service basics in *Utility Tools*.

Using the Web Service Pageflow Wizard

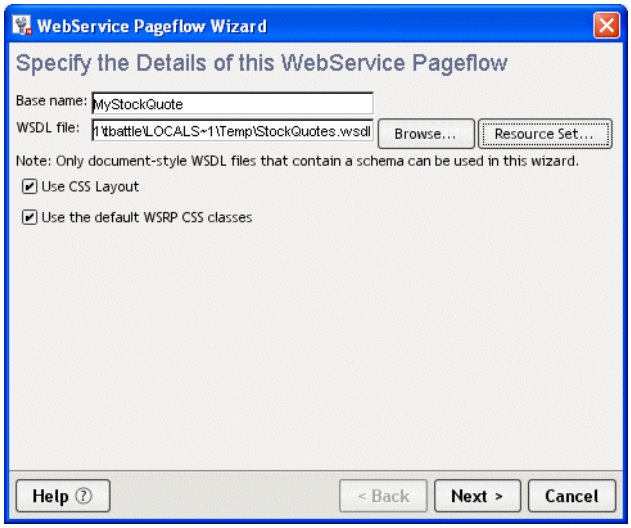
➢ **To create a Web Service pageflow process:**

1. With your project open in exteNd Director, select **File>New**.
2. Click the **Portlet** tab.
3. Select **Web Service Pageflow** and click **OK**:

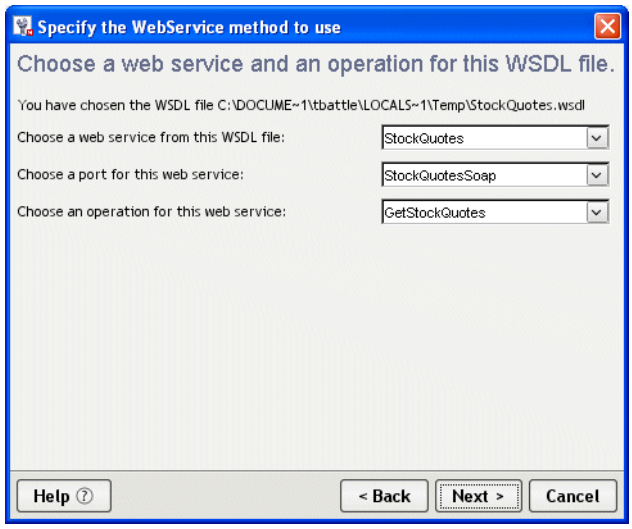


4. Type a base name for the pageflow in the **Base name** field. This name is used as a prefix for the files that will be generated.
5. In the **WSDL file** field, specify a path to the WSDL file that describes the service.

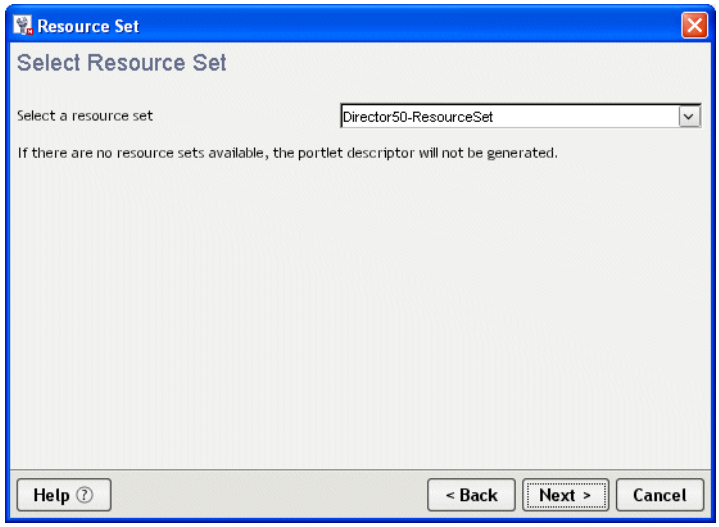
   If the file is located within the resource set, you can navigate to it by clicking the Resource Set button. If the file is located on the file system, you can find it by clicking the Browse button:



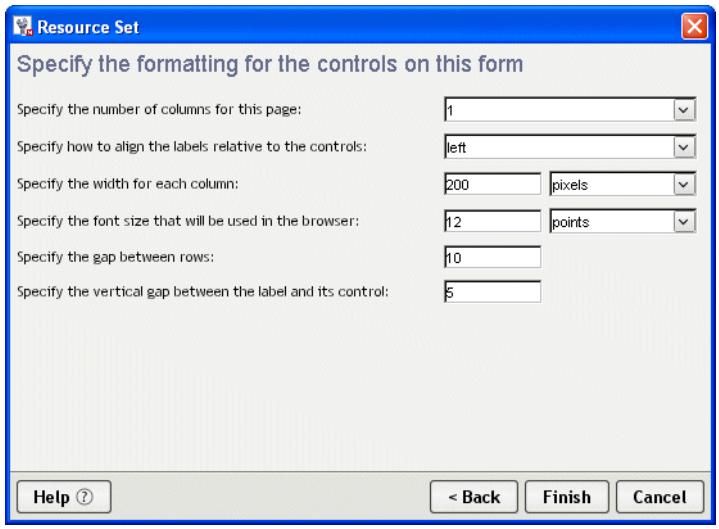
6. Click **Next**.
7. Choose a Web Service, a port, and an operation and click **Next**:



8 Select the target resource set in the **Select a resource set** dropdown and click **Next**:



9 Specify how you want to format the controls on the generated forms as follows:



10 Click **Finish**.

9 Composer Pageflow Wizard

This chapter provides instructions for using the Composer Pageflow Wizard. It contains the following sections:

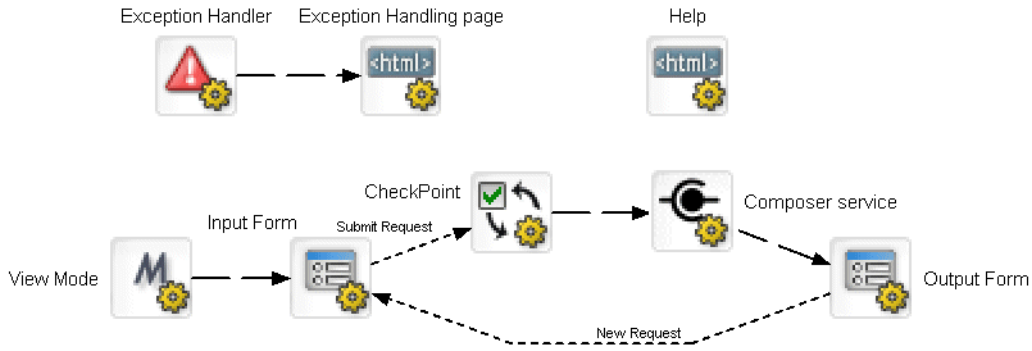
About the Composer Pageflow Wizard

exteNd Director provides the **Composer Pageflow Wizard** to help you create pageflows that execute exteNd Composer services.

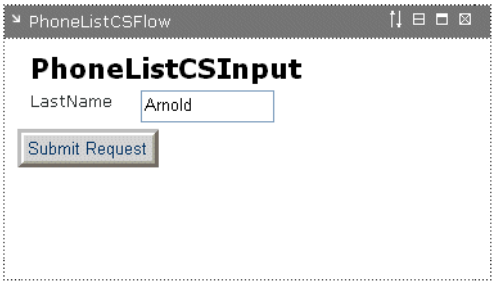
After you use the wizard, you can run the generated pageflow right away. You can also use the Pageflow Modeler to make changes to the pageflow, just as you would with any other pageflow.

The Composer Pageflow Wizard generates a set of forms (XHTML pages that use XForms technology) as well as a pageflow process that ties the exteNd Composer service and the forms together into a simple application. This application provides a convenient and easy-to-use interface for invoking an exteNd Composer service.

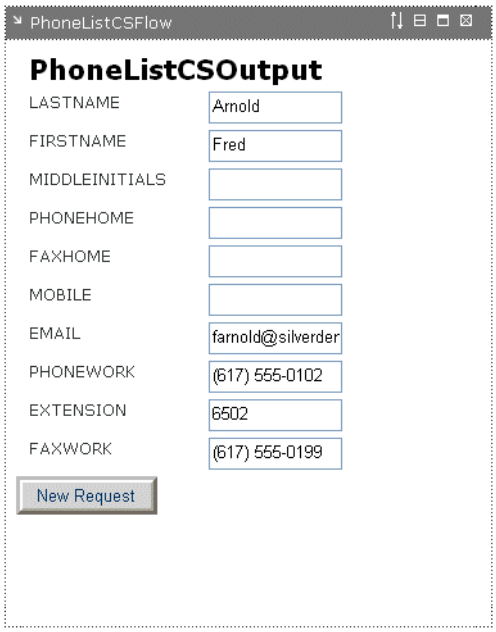
Example The following example shows a typical exteNd Composer pageflow:



At runtime this pageflow would display an input form that allows the user to pass parameters to the service:



When the user submits the form, the service executes. When the service has finished processing, the pageflow displays an output form that shows the data returned by the service.



This particular service uses a JDBC component to connect to a database and retrieve data from a table that contains a list of employees and phone numbers.

A typical exteNd Composer pageflow also includes a common help page, as well as an exception handler.

Adding an exteNd Composer project

Before you run the Composer Pageflow Wizard, you need to add an exteNd Composer subproject to your exteNd Director project. You can do this in either of two ways:

- By creating a new exteNd Composer project
- By adding an existing exteNd Composer project

📖 For details on adding exteNd Composer subprojects, see the chapter on working with exteNd Composer projects.

Deploying the project

Once you've added the exteNd Composer subproject to your exteNd Director project, you need to deploy your exteNd Director project at least once. If you do not deploy the project, the portlet running the pageflow will most likely display a `java.lang.IllegalStateException` that indicates that the service could not be initialized.

After deploying the project, you can begin to take advantage of the vulturing capabilities provided by the resource set, since the exteNd Composer subproject is added to the resource set. The resource set ensures that any change you make to an exteNd Composer project artifact is automatically picked up by the server and can be tested right away. This is also true of the pageflow, since pageflows are also stored in the resource set as well.

NOTE: The resource set that is being used by exteNd Composer must be the same as the one used by the pageflow. If they are not the same, the pageflow will not be able to find all of the resources associated with the service.

 For complete details on working with exteNd Composer services, see the exteNd Composer Help.

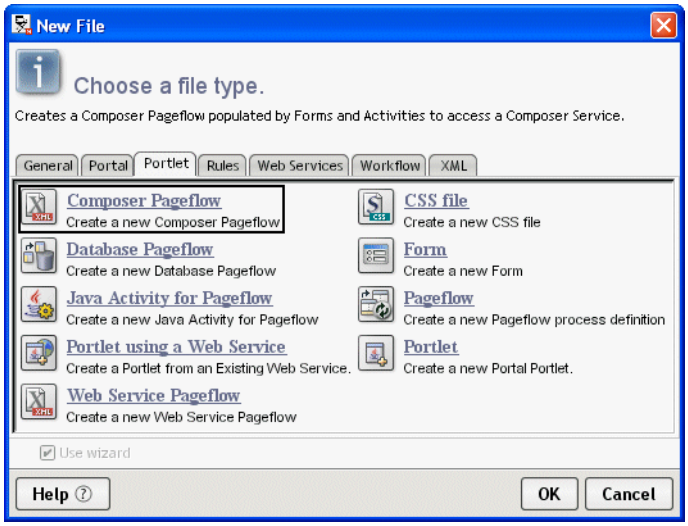
Using the Composer Pageflow Wizard

➢ **To create an exteNd Composer pageflow process:**

1 With your project open in exteNd Director, select **File>New**.

2 Click the **Portlet** tab.

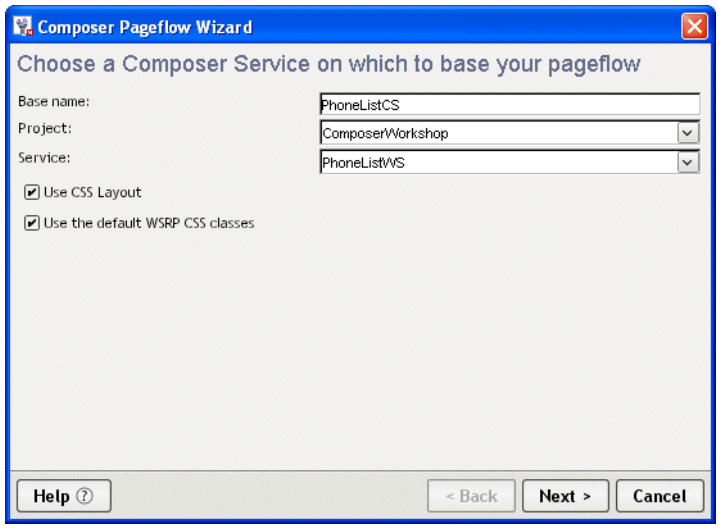
3 Select **Composer Pageflow** and click **OK**:



4 Type a base name for the pageflow in the **Base name** field. This name is used as a prefix for the files that will be generated.

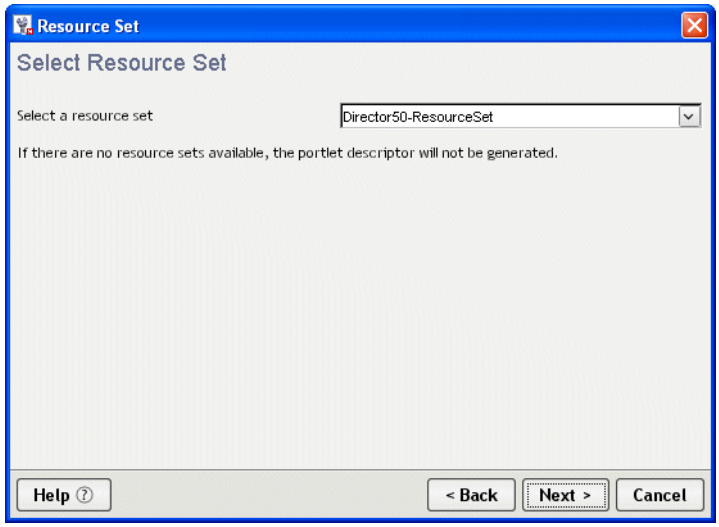
5 In the **Project** dropdown, select the exteNd Composer project that contains the service you want to execute.

6 In the **Service** dropdown, select the service to execute:

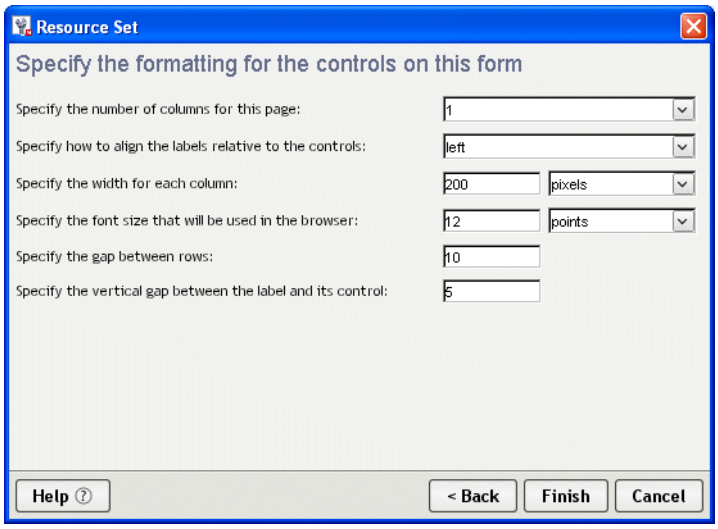


7 Click **Next**.

8 Select the target resource set in the **Select a resource set** dropdown and click **Next**:



9 Specify how you want to format the controls on the generated forms as follows:



10 Click **Finish**.

10 Java Activity Wizard

This chapter describes how to create a Java activity to use in a pageflow process. It has these sections:

About Java activities

A **Java activity** is a system activity that executes a Java class within the context of a pageflow. A Java activity allows you to write custom business logic that executes automatically without user intervention.

You can create the Java activity using the Java Activity Wizard, code the resulting Java class template, and add the activity in the Pageflow Modeler. The pageflow engine automatically forwards work after the Java Activity is processed.

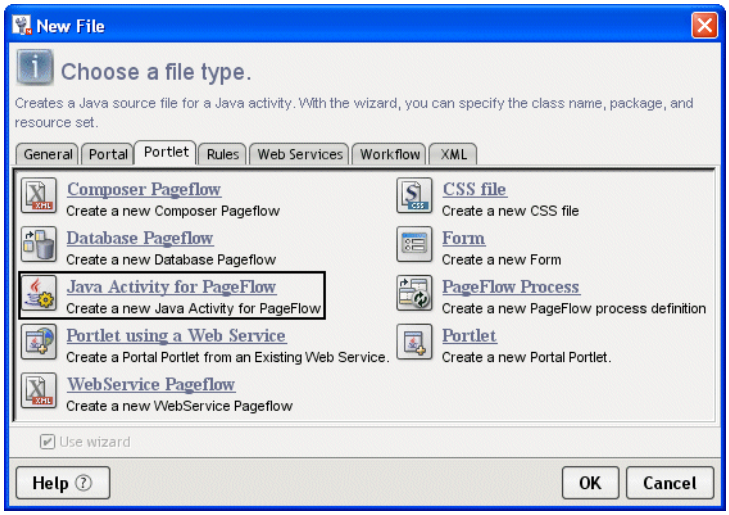
For information about using the activity within your pageflow, see "Java activity" on page 48.

Using the Java Activity Wizard

➢ **To generate the Java Activity code template:**

1 With your project open in exteNd Director, select **File>New>File**.

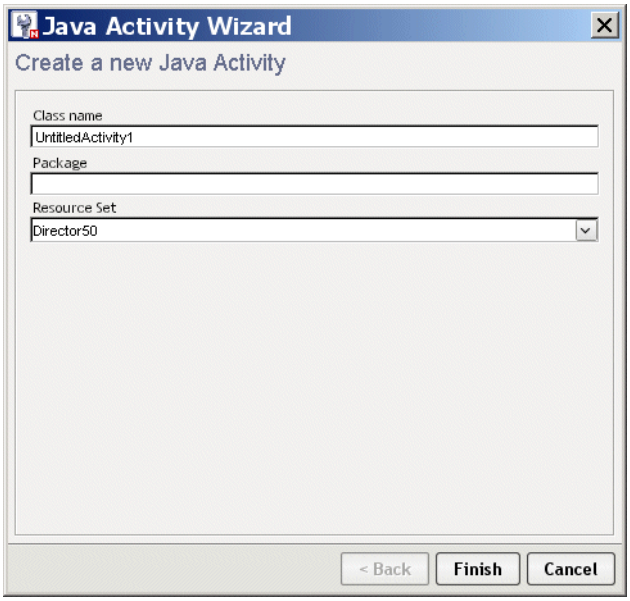
2 From the New File dialog select the **Portlet** tab and click **Java Activity for Pageflow**:



3 Click OK to start the wizard.

4 Complete the wizard panel:

Option	What to do
Class name	Specify a class name for the Java activity
Package	(Optional) Specify a package hierarchy (with levels separated by periods) to place the Java activity in a subdirectory of the base directory. This affects only the directory where the Java activity is saved. For example, if the base directory is *ProjectDir*/src and you specify **com.myco** as the package, the Java activity will be created in *ProjectDir*/src/com/myco.
ResourceSet	Select the Resource Set in which to store your application data. 📖 For more information, see the chapter on using the resource set in an exteNd Director application in *Developing exteNd Director Applications*.



5 Click **Finish**. The wizard generates the Java source template. Click **OK** on the popup to access the template.

Coding the Java activity

The generated class implements the EbiJavaActivity interface and generates a method stub for the invoke() method. This method supplies the context, and is called when work is forwarded to the Java activity in the pageflow process.

Accessing a scoped path from a Java activity

The following example shows how to code the invoke() method to access a scoped path that is defined in the pageflow in which the Java activity is running:

```
import com.sssw.wf.api.*;

  public void invoke(EbiContext context) {
        /*
        try {
```

```
        // how to get a value from a scopedPath ( assuming a request var of fname )
        com.sssw.fw.api.EbiScopedPath fname =
          com.sssw.fw.factory.EboScopedPathFactory.createScopedPath(
             "/Request/param/fname");
        String theFirstName = (String)fname.getValue( context );

        // how to set a value on a scopedPath.
        com.sssw.fw.api.EbiScopedPath sessionDoc =
          com.sssw.fw.factory.EboScopedPathFactory.createScopedPath(
            "/Session/DOC");
        sessionDoc.setValue( context, "mySessionDocValue" );

        // how to copy the request Referer into a session variable
        com.sssw.fw.api.EbiScopedPath from =
          com.sssw.fw.factory.EboScopedPathFactory.createScopedPath(
            "/Request/prop/Referer");
        com.sssw.fw.api.EbiScopedPath to =
          com.sssw.fw.factory.EboScopedPathFactory.createScopedPath(
            "/Session/Referer");
        com.sssw.fw.core.EboScopedPathUtil.copy( context, from, to );
    }
    catch (Exception e) {
        System.out.println(e);
    }
    */

}
```

Performing a JNDI lookup

In the code for a Java activity, you cannot use the InitialContext object to perform a JNDI lookup. Instead, you need to use EbiServiceLocator interface.

For example, suppose you have an environment entry in the web.xml descriptor for a exteNd Director WAR file:

```
<env-entry>
<env-entry-name>mydata</env-entry-name>
<env-entry-value>myvalue</env-entry-value>
<env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

If you use the following code to try to retrieve the environment entry, you will not be able to access the environment entry:

```
Context ic = new InitialContext();
Context env = (Context) ic.lookup("java:comp/env");
String value = (String) env.lookup("mydata");
```

However, the following code will work:

```
com.sssw.fw.api.EbiServiceLocator locator =
  com.sssw.fw.factory.EboFactory.getServiceLocator();
String value = (String) locator.getEnvEntry("mydata");
```

Example: Starting a workflow process

One reason you might want to execute a Java class within the context of a pageflow is to start a workflow process.

The following example shows how this is done:

```
// in the Java activity's invoke method
```

```
public void invoke(com.sssw.wf.api.EbiContext context) {
  com.sssw.wf.api.EbiContext newWFContext =
com.sssw.wf.client.EboFactory.createEbiContext();
  EbiWorkflowEngineDelegate engineDelegate =
com.sssw.wf.client.EboFactory.getWorkflowEngineDelegate();
  engineDelegate.startProcessByName("someProcess", newWFContext);
}
```

III Reference

Provides reference information describing how to use an RPC-style Web Service in a pageflow

- Chapter 11, “Working with RPC-Style Web Services”

11 Working with RPC-Style Web Services

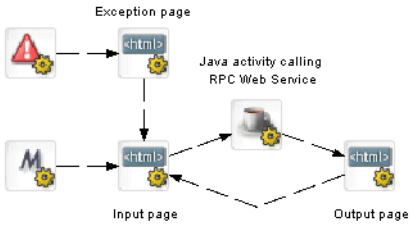
This chapter explains how to create a pageflow that invokes an RPC-style Web Service. It includes these topics:

About pageflows that use RPC-style Web Services

exteNd Director provides the **Web Service Pageflow Wizard** to help you create pageflows that execute Web Services. This wizard uses a Web Service activity to invoke the service.

The Web Service activity only provides support for document-style WSDL files. However, you can create a pageflow that invokes an RPC-style Web Service by using a Java activity.

Example Here's an example of a pageflow that executes an RPC-style Web Service:



Here's what the input page would look like at runtime:

Here's what the output page would look like at runtime

Creating a pageflow that uses an RPC-style Web Service

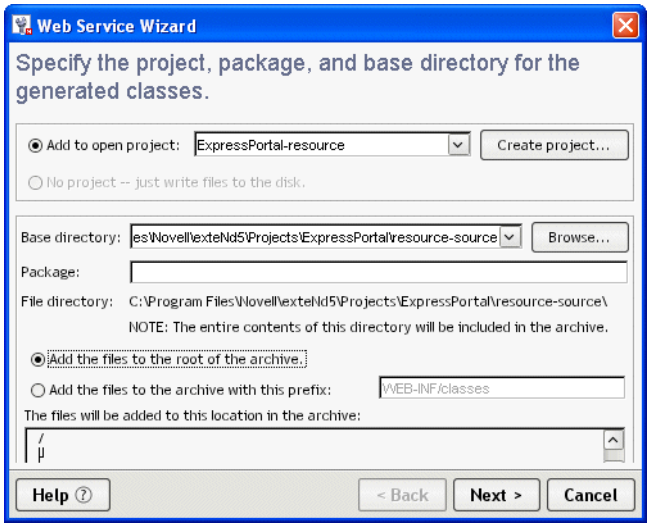
To use an RPC-style Web Service in a pageflow, you need to:

- Generate a Web Service consumer (a program that accesses a Web Service)
- Write a Java activity class that acts as a client to the Web Service
- Create the user interface for the pageflow
- Create a pageflow that includes the Java activity and the activities required to present the user interface

Generating the Web Service consumer

To generate a Web Service consumer, you need to run the **Web Service Wizard**. When you run the wizard, you provide a WSDL file as input. In this case, the WSDL file describes a Web Service that uses RPC-style bindings.

Specifying the project, package, and directory for the consumer To ensure the generated classes can be found at runtime, you need to provide the following settings when you're running the wizard:



For complete details on how to generate a consumer, see the chapter on generating Web Service consumers in *Utility Tools*.

After running the wizard, you can test the Web Service consumer by running the **Web Service Client Runner**. However, before you can do this, you need to:

- Edit the *xxx*Client.java file
- Add the archives required by the Web Services SDK to your project

Editing the xxxClient.java file Before using the generated xxxClient.java file, you:

- Must edit the process() method to call one or more methods of the target Web Service.
- May need to edit the getRemote() method to specify the correct location (binding) for accessing the target Web Service.

Adding the archives required by the Web Services SDK to your project For a complete list of the archives required, see the chapter on generating Web Service consumers in *Utility Tools*.

Writing the Java activity class

A **Java activity** is a system activity that executes a Java class within the context of a pageflow. A Java activity allows you to write custom business logic that executes automatically without user intervention.

You can create a Java activity using the Java Activity Wizard, code the resulting Java class template, and add the activity in the Pageflow Modeler.

To invoke a Web Service from a Java activity, you need to include some Java code to:

- Create the initial context
- Perform a lookup
- Get the port for the service
- Invoke the Web Service method
- Using code generated by the Web Service Wizard
- This logic is generated for you when you run the Web Service Wizard. You can simply copy this logic from the generated *xxx*Client.java file to the invoke() method on your Java activity, as shown below:

```
import com.sssw.wf.api.*;
import javax.naming.*;
import net.xmethods.sd.*;

public class TemperatureJavaActivity implements EbiJavaActivity
    public TemperatureJavaActivity() {
    }

    public void invoke(EbiContext context) {

        System.out.println("TemperatureJavaActivity");
        try {

            InitialContext ctx = new InitialContext();
             String lookup = "xmlrpc:soap:net.xmethods.sd.TemperatureService";
            TemperatureService service = (TemperatureService)ctx.lookup(lookup);
            TemperaturePortType remote =
(TemperaturePortType)service.getTemperaturePort();
            // get the ZIP from the portlet session
            String zip = (String)context.getEbiWhiteboard().getScopedValue( "ZIP"
);
            float f = remote.getTemp(zip);
            context.getEbiWhiteboard().setScopedValue("TEMP", String.valueOf( f )
);

        }
        catch (Exception e) {
            throw new RuntimeException (e);
        }

...
}
```

Note that this Java activity also uses the scoped path API to get the zip code (ZIP) and set the temperature (TEMP) on the Session object.

Using a helper class to invoke the service The exteNd Director API provides a helper class you can use to invoke a Web Service. This class is called **WebServiceActivityHelper**. It is located in the **com.novell.afw.portal.portlet.pf** package. The code example below shows the use of this helper class:

```
import com.sssw.wf.api.*;
import com.novell.afw.portal.portlet.pf.*;

public class TemperatureJavaActivity implements EbiJavaActivity {
    public TemperatureJavaActivity() {
    }
```

```
    public void invoke(EbiContext context) {

        System.out.println("TemperatureJavaActivity");
        try {

            WebServiceActivityHelper wsHelper
                = new WebServiceActivityHelper();
            net.xmethods.sd.TemperaturePortType remote =
                (net.xmethods.sd.TemperaturePortType)
                wsHelper.getRemote(
                 net.xmethods.sd.TemperatureService.class,
                 net.xmethods.sd.TemperaturePortType.class);

            // get the ZIP from the portlet session
            String zip =
             (String)context.getEbiWhiteboard().getScopedValue( "ZIP" );
            float f = remote.getTemp(zip);
            context.getEbiWhiteboard().setScopedValue(
             "TEMP", String.valueOf( f ) );

        }
        catch (Exception e) {
            throw new RuntimeException (e);
        }

...
}
```

Creating the user interface for the pageflow

Typically, you'll want to create an input page as well as an output page for a pageflow that invokes a Web Service. The input page allows the user to enter values for parameters that should be passed to the service. The output page displays the data returned by the service.

Input page

Here's some HTML for a sample input page:

```
<form name="form1" method="post" action="wsrp_rewrite?wsrp-
urlType=blockingAction/wsrp_rewrite">
<br/>
    <span class="portlet-form-field-label">
        Zip
    </span>
    <input class="portlet-form-field" type="name" name="zip" value="02630"
size="20”>
    <br/><br/>
    <input type="submit" name="verb" value="Continue">
</form>
```

Output page

Here's some HTML for a sample output page:

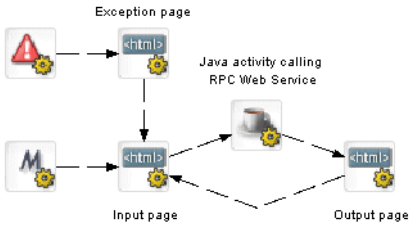
```
<form name="form1" method="post" action="wsrp_rewrite?wsrp-
urlType=blockingAction/wsrp_rewrite">
<br/>
    <span class="portlet-form-field-label">
        Zip Temperature : scopedpath?Session/TEMP/scopedpath
    </span>
    <br/><br/>
    <input type="submit" name="verb" value="Continue">
</form>
```

In this example, the data returned (the temperature) is retrieved by means of a scoped path expression:

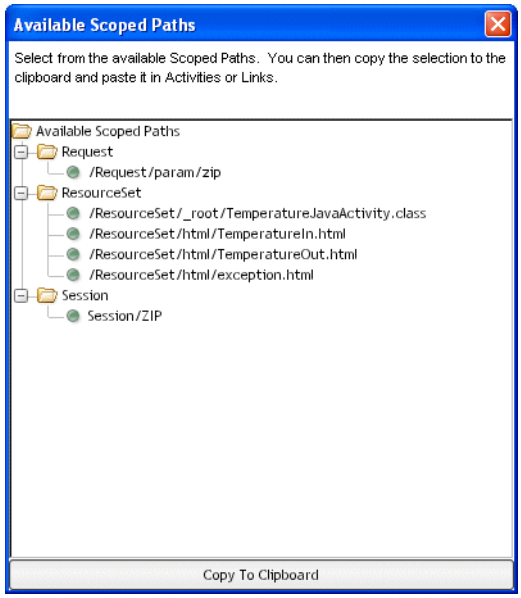
```
scopedpath?Session/TEMP/scopedpath
```

Creating the pageflow

Here's what the pageflow would look like:

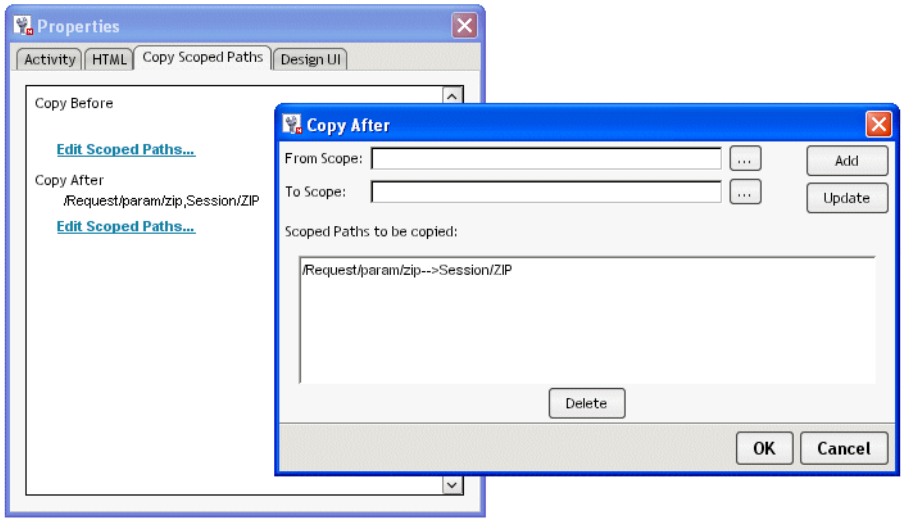


Here are the scoped paths for the flow:



Specifying a Copy After operation for the input page

The HTML input page would specify a Copy After operation that copies the zip code entered by the user into a variable on the Session scope:


Properties
Activity
HTML
Copy Scoped Paths
Design UI
Copy Before
Edit Scoped Paths...
Copy After
/Request/param/zip,Session/ZIP
Edit Scoped Paths...
Copy After
From Scope:
To Scope:
Add
Update
Scoped Paths to be copied:
/Request/param/zip-->Session/ZIP
Delete
OK
Cancel

Index

E

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W

X

Z