

Novell Integration Manager™

www.novell.com

TUTORIAL



Novell®

Legal Notices

Copyright ©1999, 2000, 2001, 2002, 2003, 2004, 2005 Novell, Inc. and SilverStream Software, LLC. All rights reserved.

SilverStream software products are copyrighted and all rights are reserved by SilverStream Software, LLC

SilverStream is a registered trademark of SilverStream Software, LLC. Novell is a registered trademark of Novell, Inc.

Title to the Software and its documentation, and patents, copyrights and all other property rights applicable thereto, shall at all times remain solely and exclusively with SilverStream and its licensors, and you shall not take any action inconsistent with such title. The Software is protected by copyright laws and international treaty provisions. You shall not remove any copyright notices or other proprietary notices from the Software or its documentation, and you must reproduce such notices on all copies or extracts of the Software or its documentation. You do not acquire any rights of ownership in the Software.

Jakarta-Regexp Copyright ©1999 The Apache Software Foundation. All rights reserved. Ant Copyright ©1999 The Apache Software Foundation. All rights reserved. Xalan Copyright ©1999 The Apache Software Foundation. All rights reserved. Xerces Copyright ©1999-2000 The Apache Software Foundation. All rights reserved. Jakarta-Regexp, Ant, Xalan, Crimson and Xerces software is licensed by The Apache Software Foundation and redistribution and use of Jakarta-Regexp, Ant, Xalan, Crimson and Xerces 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 notices, 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 "The Jakarta Project", "Jakarta-Regexp", "Xerces", "Xalan", "Ant" 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 <<mailto: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.

Copyright ©2000 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 <<mailto: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 <<mailto:pm@jdom.org>>). 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.

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.

Copyright ©2001 Extreme! Lab, Indiana University License. <http://www.extreme.indiana.edu>. Permission is hereby granted, free of charge, to any person obtaining a copy of the Indiana University software and associated Indiana University documentation files (the "IU Software"), to deal in the IU Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the IU Software, and to permit persons to whom the IU Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the IU Software. THE IU SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE IU SOFTWARE OR THE USE OR OTHER DEALINGS IN THE IU SOFTWARE.

This Software is derived in part from the SSLava™ Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved.

Copyright © 1994-2002 W3C® (Massachusetts Institute of Technology, Institut National de Recherche Informatique et en Automatique, Keio University), all Rights Reserved. <http://www.w3.org/consortium/legal>. This W3C work (including software, documents, 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 use, 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, that you make: 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, a short notice of the following form (hypertext is preferred, text is permitted) should be used within the body of any redistributed or derivative code: "Copyright © [Date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>" 3. Notice of any changes or modifications to the W3C 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.

Novell, Inc.
1800 South Novell Place
Provo, UT 85606

www.novell.com

Novell Integration Manager Tutorial
May 2006

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

Contents

About This Guide	5
1 Introduction to the Integration Manager Environment	7
What you will learn	7
What you will do	7
Tutorial Setup	8
MySQL	8
JBoss	8
Integration Manager Basics	9
Integration Manager Building Blocks	9
Launching Integration Manager	10
EXERCISE 1-1: Launch Integration Manager	10
Navigating in the Integration Manager Environment	11
EXERCISE 1-2: Explore the Integration Manager Environment	11
Integration Manager Project Structure	12
About the Tutorial Project	12
Project Requirements	14
EXERCISE 1-3: Create a Integration Manager Project	14
Summary of what you've done	16
2 Components and XML Mapping	17
What you will learn	17
What you will do	17
Design and Structure of Integration Manager Components	18
Integration Manager Component Architecture	18
Document Object Model (DOM)	18
XML Map Action	19
Component Animation	19
EXERCISE 2-1: Create an XML Map Component	20
EXERCISE 2-2: Add a Map Action to the Component	21
EXERCISE 2-3: Modify the Input DOM and Save as an XML Template	22
EXERCISE 2-4: Animate the Map Component	24
Summary of what you've done	25
3 XML Templates	27
What you will learn	27
What you will do	27
Integration Manager XML Templates	28
XML Template Categories	28
XML Sample Documents	28
XML Schema	28
XSL Style Sheets	29
XML Templates	29

	EXERCISE 3-1: Work with XML Templates	29
	EXERCISE 3-2: Create an XML Template using the template wizard	30
	EXERCISE 3-3: Import an XML template from another Integration Manager project	32
	EXERCISE 3-4: Edit the PatientLookup XML Map Component to use the XML Templates	34
	Summary of what you've done	35
4	Web Services	37
	What you will learn	37
	What you will do	37
	Integration Manager Web Services	38
	Web Service Definition	38
	Services vs. Components	38
	EXERCISE 4-1: Create a Web Service	38
	Summary of what you've done	42
5	Deployment	43
	What you will learn	43
	What you will do	43
	Integration Manager Web Service Deployment	44
	Deployment Considerations	44
	Server Profile	45
	EXERCISE 5-1: Creating a Server Profile	45
	Deployment xObject	47
	EXERCISE 5-2: Create a Deployment xObject	48
	Service Triggers	52
	EXERCISE 5-3: Set up the Service Trigger(s) for your Web Service	52
	EXERCISE 5-4: Import the MySQL JAR resource file	53
	EXERCISE 5-5: Deploying to the JBoss application server	55
	Summary of what you've done	58
6	JDBC and LDAP	59
	What you will learn	59
	What you will do	59
	Integration Manager Connection Resources and Connection Components	60
	JDBC Connection	60
	EXERCISE 6-1: Create a JDBC Connection	60
	JDBC Component	63
	EXERCISE 6-2: Create a JDBC Component	64
	EXERCISE 6-3: Access a Database using the JDBC Component	66
	LDAP Connection	69
	EXERCISE 6-4: Check for an active LDAP directory connection	70
	EXERCISE 6-5: Start the Novell exteNd LDAP Utility	70
	EXERCISE 6-6: Create an LDAP Connection	71
	LDAP Component	73
	EXERCISE 6-7: Create an LDAP Component	73
	EXERCISE 6-8: Add a Create DSML action to the Action Model	76
	EXERCISE 6-9: Set the Depth of the Directory Search	77

EXERCISE 6-10: Create the Filter for the Search	79
EXERCISE 6-11: Select the attributes to be returned by the search	80
EXERCISE 6-12: Add the Execute DSML action to the Action Model	80
Summary of what you've done	82
7 Basic Integration Manager Actions	83
What you will learn	83
What you will do	83
Integration Manager Action Model and Actions	84
Comment Action	84
Decision	85
Log Action	85
Component	85
Function	86
XPath and ECMAScript (The Integration Manager Expression Builder)	86
EXERCISE 7-1: Set up the component for edits	87
EXERCISE 7-2: Add a comment to the Action Model	88
EXERCISE 7-3: Add a Decision Action	89
EXERCISE 7-4: Add Map Actions	91
EXERCISE 7-5: Add a Log Action	92
EXERCISE 7-6: Create or increment the hit count for the physician	94
EXERCISE 7-7: Add Actions to the False Branch of the Decision action	96
EXERCISE 7-8: Add Actions to the True Branch of the Decision action	100
EXERCISE 7-9: Edit the XML map component properties	105
EXERCISE 7-10: Add Component Actions	106
EXERCISE 7-11: Add Map Actions	108
Summary of what you've done	110
8 Publishing and Consuming Web Services	111
What you will learn	111
What you will do	111
Publishing Web Services	112
Servlets	112
Web Service Description Language (WSDL)	112
Consuming Web Services	113
Java Server Pages (JSP)	113
XForms	113
XML Interchange	114
WS Interchange	114
EXERCISE 8-1: Create a JSP resource file	114
EXERCISE 8-2: Add the JSP to the Deployment Component	116
EXERCISE 8-3: Deploy the Web Service with the JSP	117
EXERCISE 8-4: Create an XML Map Component that uses XML interchange	117
EXERCISE 8-5: Modify the Deployment Component for the XML Interchange	119
EXERCISE 8-6: Deploy the Web Service with Servlet Type XML(HTTP/Post)	120
EXERCISE 8-7: Create a WSDL for the PatientRecResponse Web Service	120
EXERCISE 8-8: Create XML Templates for the WSDL	122

EXERCISE 8-9: Add a Soap Trigger to the Deployment Component	123
EXERCISE 8-10: Deploy the Web Service with SOAP	124
EXERCISE 8-11: Create an XML Map component that executes a WSInterchange action	124
EXERCISE 8-12: Create an XForm for Physician Input.	127
EXERCISE 8-13: Create a Web Service that uses the XForm	130
EXERCISE 8-14: Create the deployment profile for the XFormWS.....	132
EXERCISE 8-15: Deploy the XForm Web Service	134
Summary of what you've done	135

About This Guide

Purpose

This tutorial introduces you to the environment and features of Novell Integration Manager. You will learn about:

- ◆ The Integration Manager design-time environment
- ◆ XML Templates
- ◆ Integration Manager Actions and the Action Model
- ◆ Integration Manager Connection Resources
- ◆ Deploying Web Services with Integration Manager

Audience

This tutorial is intended for application developers who will be using Integration Manager to develop J2EE applications, especially XML integration applications and Web Services.

Prerequisites

Experience: No prior Java programming experience is required to complete the tutorial. It is helpful to have some knowledge of the following:

- ◆ J2EE file packaging concepts (EAR, WAR, JAR)
- ◆ XML and its related standards (XSL, XPath, Schema)

Software: This tutorial is designed to use the following software. (All of these are included except for MySQL and the JBoss application server.)

- ◆ Novell Integration Manager
- ◆ JBoss application server 4.0.3 (download from <http://labs.jboss.com/portal/download>)
- ◆ MySQL (download from <http://www.mysql.com>)
- ◆ Novell exteNd LDAP Utility

- ◆ A web browser that supports HTML 4.0 and (optionally) XML+XSL, to see the final output of the service you deploy

Organization

A summary of the lessons you'll find in this tutorial:

Lesson	Description
1	Lesson 1, "Introduction to the Integration Manager Environment"
2	Lesson 2, "Components and XML Mapping"
3	Lesson 3, "XML Templates"
4	Lesson 4, "Web Services"
5	Lesson 5, "Deployment"
6	Lesson 6, "JDBC and LDAP"
7	Lesson 7, "Basic Integration Manager Actions"
8	LESSON 8, "Publishing and Consuming Web Services"

Additional documentation

For the complete set of [Novell Integration Manager documentation](http://www.novell.com/documentation-index/index.jsp), see the [Novell Documentation Web Site \(http://www.novell.com/documentation-index/index.jsp\)](http://www.novell.com/documentation-index/index.jsp).

1

Introduction to the Integration Manager Environment

What you will learn

This lesson describes the Novell Integration Manager environment, and shows you how to use Integration Manager's basic UI features. You will be introduced to some of the nomenclature associated with the creation of web services, and learn about Integration Manager's *project* architecture.

You will learn about Integration Manager Basics:

- ◆ Integration Manager Building Blocks
- ◆ Launching Integration Manager
- ◆ Navigating in the Integration Manager Environment
- ◆ Integration Manager Project Structure

What you will do

You will perform the following tasks using Integration Manager:

- 1 Launch Integration Manager
- 2 Explore the Integration Manager Environment
- 3 Create a Integration Manager Project

How long will it take? About 10 minutes

Tutorial Setup

MySQL

This tutorial requires the MySQL database manager, version 4.1 or higher. MySQL is not included with Novell Integration Manager. You can download MySQL from <http://dev.mysql.com/downloads/>. When you install MySQL, configure MySQL to use port 63306.

Once MySQL is installed, you need to create a database named “samples50” in the MySQL \data directory. The Integration Manager installation includes a SQL script file that you can run to automatically create the samples50 database and create the required table structure. This file is named samples50.sql, and is located in \novell\nim6\Composer\tutorial\files. Follow these steps to create the samples50 database:

- 1 From the mysql prompt, type the following command:

```
source <install  
path>\nim6\Composer\tutorial\files\samples50.sql;
```

Replace <install path> with the path to your Integration Manager installation. For example, if you installed Integration Manager in the default location, the command would be:

```
source C:\novell\nim6\Composer\tutorial\files\samples50.sql;
```

MySQL displays output messages that indicate that the database has been created.

- 2 Type “quit;”, then press the Enter key to exit from MySQL.

JBoss

NOTE: This tutorial requires the JBoss application server, version 4.0.3 or higher. JBoss is not included with Novell Integration Manager. You can download JBoss from <http://labs.jboss.com/portal/download>. You must install the JBoss application server, and install the Integration Manager Enterprise server to JBoss (see the *Integration Manager Installation Guide*) before you can use this tutorial.

Once you have installed MySQL and created the samples50 database, you need to create a datasource for JBoss, and install the MySQL JDBC driver into JBoss.

- 1 Edit the `mysql-ds.xml` file located in <JBoss install directory>\docs\examples\jca.
 - ◆ Replace <jndi-name>MySqlDS</jndi-name> with <jndi-name>samples50</jndi-name>.

- ◆ Replace `<connection-url>jdbc:mysql://mysql-hostname:63306/jbossdb</connection-url>` with your connection string—that is, your mysql hostname (for example, “localhost”) and port number (63306 is the default MySQL port number when installed with Integration Manager). For example, `jdbc:mysql://localhost:63306/samples50`.
 - ◆ Set the **user-name** and **password** elements to your database username and password.
 - ◆ Save the `mysql-ds.xml` file in the **deploy** directory for the server in which the Integration Manager Enterprise Server is installed (for example, `<JBoss install directory>\server\nim\deploy`).
- 2** Copy the MySQL JDBC driver (for example, `mysql-connector-java-bin.jar`) into the JBoss server **lib** directory. For example, the default Integration Manager installation creates a JBoss server directory named “nim”. In this case, you would copy the JDBC driver to `/<JBoss install directory>/server/nim/lib`.

Integration Manager Basics

Novell Integration Manager (hereafter just “Integration Manager”) provides rich-UI design environment in which to design, create, test, debug, package, and deploy web services.

Integration Manager Building Blocks

Integration Manager wraps almost all of the objects you create in a special kind of object known as an xObject. An xObject is in most cases little more than an XML metadata description of a deployable artifact.

There are four main categories of xObject:

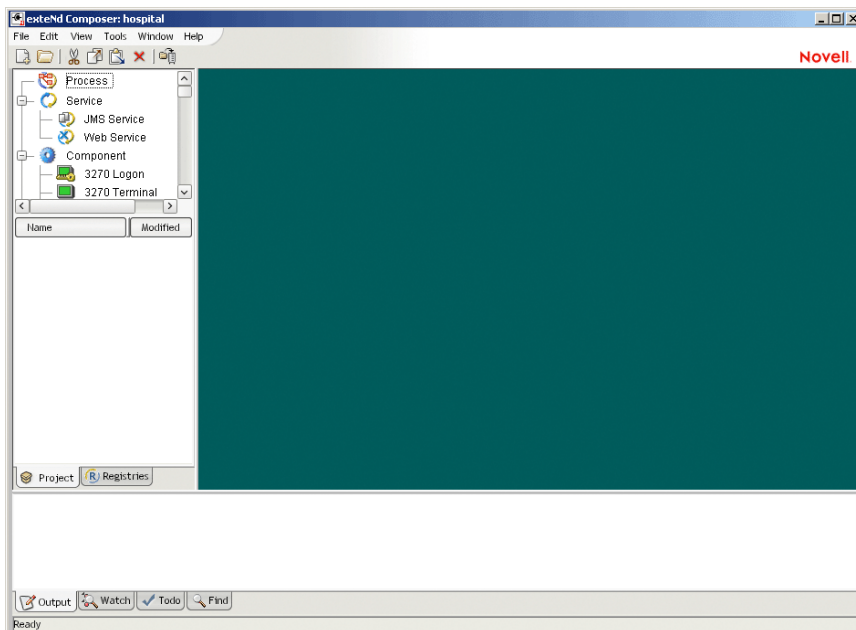
- ◆ **Services** – A service is a high level object that performs a unit of work or business transaction. Services are deployed on the application server and take XML documents as their input. Services utilize components to operate on the XML data and produce output XML documents.
- ◆ **Components** – A component in Integration Manager performs a unit of work or business logic, and is invoked by a service or another Integration Manager component. Components XML enable applications, i.e. a relational database, via the JDBC component.
- ◆ **Resources** – Components and Services utilize resource objects to perform tasks. Resources are specialized objects that perform such functions as establishing connections, map code tables, and define schemas for XML documents.
- ◆ **XML Templates** – XML Templates provide a way of categorizing and aggregating the sample XML documents your services operate on at design time.

Launching Integration Manager

When Integration Manager launches, it opens the last project that the user opened or modified. The *first* time you launch Integration Manager after installation, it will start with the tutorial project (“hospital”) as the current project. You can tell which project is the currently open project by looking at the title bar on the main Integration Manager window.

EXERCISE 1-1: Launch Integration Manager

- 1 You may launch Integration Manager in any of the following ways
 - ◆ From the Windows **Start** Menu, click **Programs>Novell Integration Manager 6.0>Integration Manager**
 - ◆ Open Windows Explorer, navigate to the **nim6\Composer\Designer\bin** directory under the location where you installed Integration Manager and double-click on the **XC.exe** file
 - ◆ Use the icon in the Novell Linux Desktop menu panel
 - ◆ (Linux) Open a terminal window and navigate to `/Novell/NIM/Composer/Designer/bin`, type the command `./xc`, then press **Enter**.
 - ◆ Upon successful launch you will see a screen similar to the one depicted here.



Navigating in the Integration Manager Environment

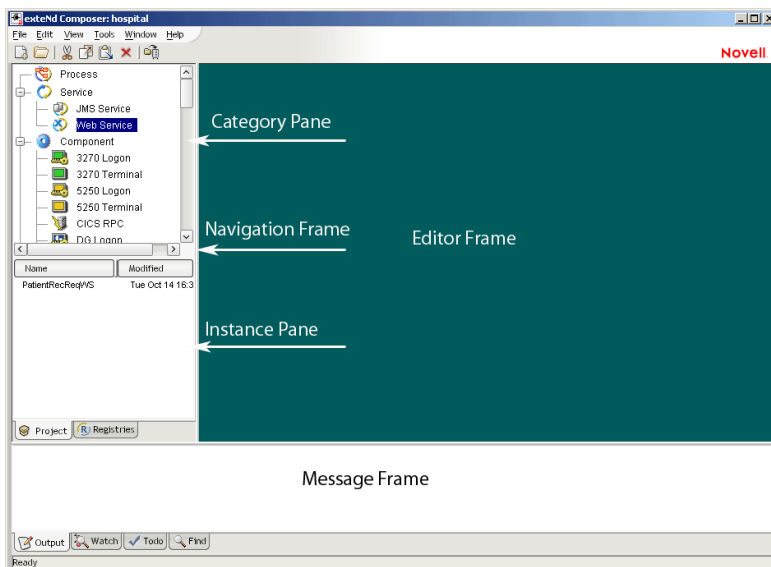
Integration Manager contains several editors and wizards to aid in the design, building, testing, and debugging of Integration Manager applications. When Integration Manager is first launched it displays the navigation, editor and message frames.

EXERCISE 1-2: Explore the Integration Manager Environment

- 1 Examine the Integration Manager Frames and become familiar with the function of the different panes.
 - ◆ **Navigation Frame** – The Navigation frame provides the user with a visual interface for creating and editing Integration Manager objects.

The upper, category pane, displays icons of the services, components and resources available to the user to build Integration Manager projects.

The lower, instance pane, displays project defined objects for the selected xObject in the category pane.
 - ◆ **Editor Frame** – Component editor content such as DOM and action models are displayed in the editor frame.
 - ◆ **Message Frame** – The message frame is composed of four tabs: the find tab displays the result of system searches. The log tab displays error and log messages. The watch tab displays system watch variables. The Todo tab allows you to track project related tasks.



In the example shown above, a project named “hospital” is the current Integration Manager project. The Web Services Component has been selected in the category pane. The instance pane shows all the web services defined in the project “hospital.”

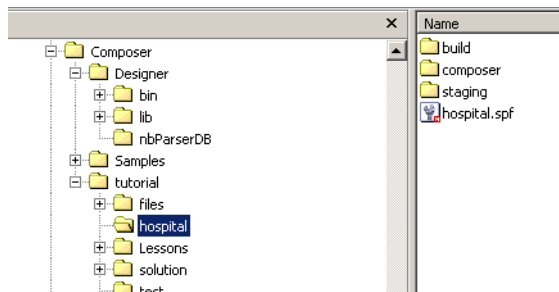
Use the mouse or keyboard and investigate the menus and options in the Integration Manager environment.

Integration Manager Project Structure

Services in Integration Manager are organized into projects. A project contains one or more deployable services and all of the code, metadata, and other resources needed by those services at runtime. Integration Manager uses projects as the unit of deployment to the application server.

When you create a project in Integration Manager a hierarchy of folders is created to organize the xObjects and aid in deployment.

The project file extension utilized by Integration Manager is **.spf**. When creating a project the **.spf** file is created in the folder you select for the project. Integration Manager creates three subdirectories under the project folder: **\Composer**, **\staging**, and **\build**. As you create your application all of the xObjects you create are stored as XML files within subdirectories named after the category of the object, i.e., **xmlcategories** under the Composer subdirectory in the project folder.



About the Tutorial Project

In the tutorial project you will be acting as both the publisher and consumer of the web service you build. The exercises in each lesson are designed to build on the previous lesson and will instruct you on how to use Integration Manager to create, deploy, and trigger the service.

Tutorial Scenario

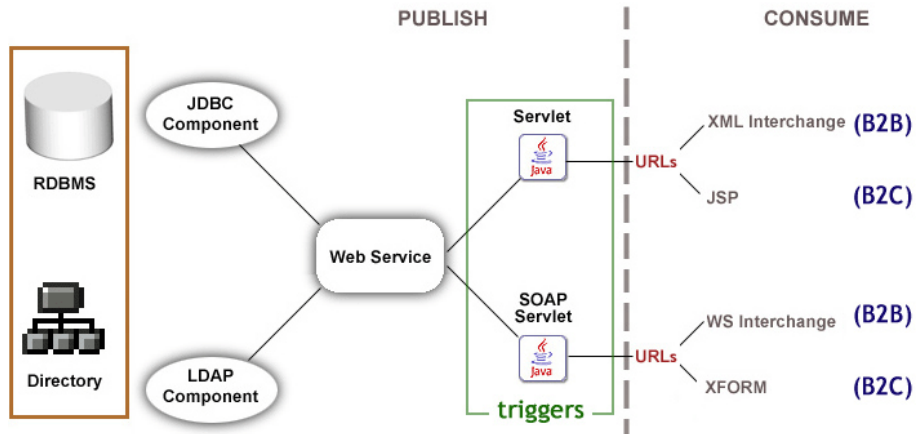
Pantagruelico Hospital maintains an e-directory of all its staff. The hospital also maintains a database of patient records. Insurance companies, medical offices and other hospital departments have a need to retrieve information from both data sources for their daily operations.

You will be creating a Integration Manager web service that will provide the client information about physicians and their patients, when supplied with a physician's name. The service will be deployed to the application server, where the client may invoke it.

The solution to the tutorial may be found under the Integration Manager install directory in the location `\Composer\tutorial\solution\hospital`.

The diagram below gives a very-high-level architectural overview of the major pieces of this tutorial. At the far left are the major data sources (a database and a directory). The Web Service wrappers a JDBC Component and an LDAP Component, which together serve as the "data access" layer of the deployed application. The service (in this tutorial, at least) is triggered by servlets, which are exposed on URLs. The triggers may (optionally) be front-ended by JSPs, XForms, or ordinary HTML (or no presentation objects at all: an RPC type of scenario, common in B2B interactions).

Tutorial Architecture



The workings of these various pieces will become clearer as you go through the lessons of the tutorial.

Project Requirements

The requirements for the service are listed below:

- 1 The service may be invoked by any of the following service triggers:
 - ◆ Servlet of the type Params(URL/Form) that outputs a raw XML document
 - ◆ Servlet of the type Params(URL/Form) that uses an XForm
 - ◆ Servlet that utilizes a JSP to format the output
 - ◆ Servlet of the type XML (HTTP/Post) that outputs an XML document
 - ◆ SOAP Servlet using WSDL
 - ◆ Servlet of the type XML(HTTP/Post) which outputs XHTML that is formatted with a stylesheet
- 2 The input will be either parameters on a URL, a submission via an XForm, or an XML document containing the physician's name.
- 3 Retrieve the patient data from the hospital database for the given physician.
- 4 Retrieve the physician's title, phone number, e-mail address, and the number of directory inquiries from the hospital e-directory.
- 5 Combine the retrieved data into an XML document and return it to the client as output.

EXERCISE 1-3: Create a Integration Manager Project

- 1 Select **File > New >Project** from the menu bar. The new project dialog will appear.
- 2 In the project name field, type: **hospital**.
- 3 Click the **Browse** button to select a Project Location.
- 4 Navigate to the **tutorial** directory located in the NIM\Composer directory where you installed the Novell software. For example D:\Program Files\Novell\NIM\Composer\tutorial.

NOTE: The location of your files is dependent on where you installed Integration Manager.



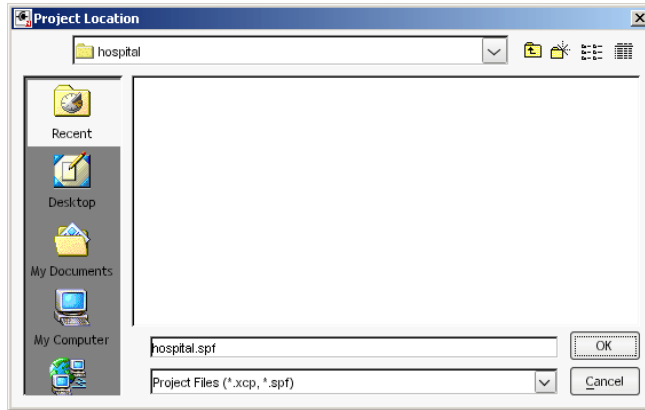
- 5 Click the **Create New Folder** button.
A folder named **New Folder** is added to the browser window. Note that the folder name is recessed and highlighted in the window.



- 6 Change the name of the folder to **hospital**

7 Click **Open**.

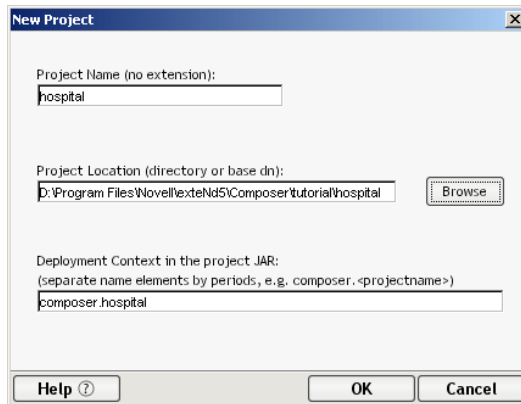
The Project Location dialog should look like the one below.



8 Click **OK** to close the project location dialog.

Accept the default deployment context, **composer.hospital** entered for you by Integration Manager in the Deployment Context textfield.

Your New Project dialog should look similar to the one below.



9 Click **OK** to close the dialog and create the project.

10 Notice that the Integration Manager title bar contains hospital as the open project.



Summary of what you've done

You have accomplished the following tasks:

- ◆ Learned about the basic Integration Manager Building Blocks.
- ◆ Launched the Integration Manager application
- ◆ Explored the Integration Manager environment
- ◆ Created a Integration Manager Project folder
- ◆ Investigated the Integration Manager project folder structure
- ◆ Created a Integration Manager Project

Next lesson

In the next lesson you will learn about DOM, Components and the XML Map Action.

2

Components and XML Mapping

What you will learn

In lesson 2 you will learn more about the design and construction of components within the Integration Manager environment. The “hospital” Service that you will create in the tutorial will execute an XML Map Component. The map component will use a Mapping Action to take input you provide and transform it to output. Within this lesson, you will create a map component that the service will utilize. This map component will take a physician’s name as input. You will use it to model the data that will be passed to your web service. Once you have your component constructed, Integration Manager’s features for animation and debugging will be introduced.

You will learn about Design and Structure of Integration Manager Components:

- ◆ Integration Manager Component Architecture
- ◆ Document Object Model (DOM)
- ◆ XML Map Action
- ◆ Component Animation

What you will do

You will perform the following exercise using Integration Manager:

- 1** Create an XML Map Component.
- 2** Add a Map Action to the Component
- 3** Modify the Input DOM and Save as an XML Template
- 4** Animate the Map Component

How long will it take? About 15 minutes

Design and Structure of Integration Manager Components

Integration Manager applications are much like any other design project. First, the service to be provided must be understood and agreed upon between the client and the service provider. Integration Manager applications are often utilized to provide services between business partners; for example, information exchange between a catalog and inventory supplier. Once the application is defined, Integration Manager components are designed, created, and packaged to implement the service.

Integration Manager Component Architecture

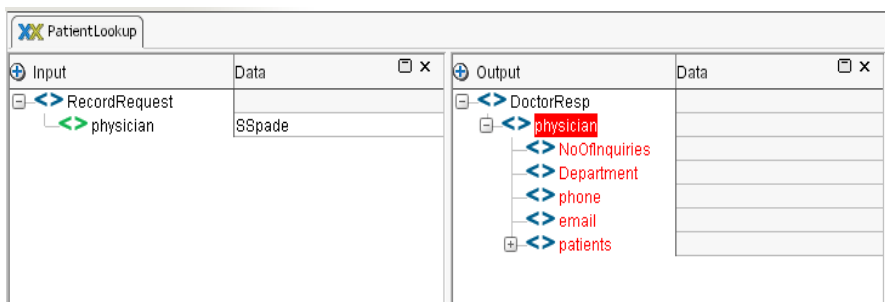
Components are the cornerstone of the Integration Manager Architecture. They use resources, connections, and actions to accomplish the requirements of a service. Since components are capable of utilizing other components, they are readily compartmentalized into units of functionality. This makes them ideal for reuse by other services and components in your design.

Components are xObjects that take XML documents as their input, operate on these documents, and produce an XML document as output.

Document Object Model (DOM)

Integration Manager represents XML documents in system memory as a DOM. The DOM represents the XML as a tree structure. The tree has a single root node, and a hierarchy of child nodes and elements that contain the data on which the component operates. The DOM provides standard methods of dealing with the XML document. A DOM is the structure that is created and manipulated at runtime.

When you create a component, Integration Manager represents the input and output XML documents as a DOM in the Integration Manager environment. The screen shot below shows a sample of a Integration Manager XML Map Component with two input XML documents and one output XML document, represented as DOMs.



XML Map Action

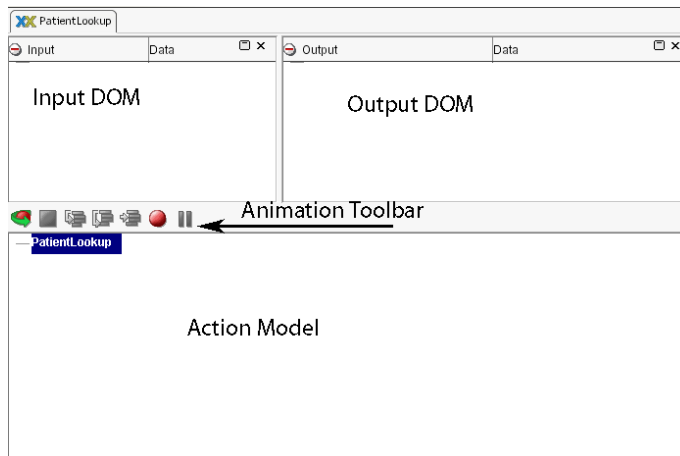
An Action in Integration Manager is similar to a programming statement. As the verb implies, something takes place when an action executes. Components utilize actions to operate on the elements and data in the XML input document to produce the required output. These actions provide a wide variety of operations, that may include transformation and re-ordering of the elements and data.

The XML Map Action is one of the most important of all the actions available within Integration Manager. A Map action can be as simple as passing the data from the input to the output with no intermediate steps, or a complex action with data transformations.

Integration Manager has two methods of creating map actions, Drag and Drop, which allows nodes from the Input DOM to be dragged and dropped in the Output DOM. The second method is via the Map Action Dialog, which provides means for more complex mappings and transformations. The Map Action Dialog also allows you to use XPath or ECMAScript expressions to address locations in the DOM. You will be using some basic XPath in the exercises within the lessons. XPath will be covered more completely later in the tutorial.

Component Animation

Integration Manager allows you to animate your components for testing and debugging. Animation provides you the ability to step into, or over, component actions as they execute. You may also set break points and watch variables to aid in the debug process. Use the animation tool bar at the top of the action model or the animate menu to access these features.

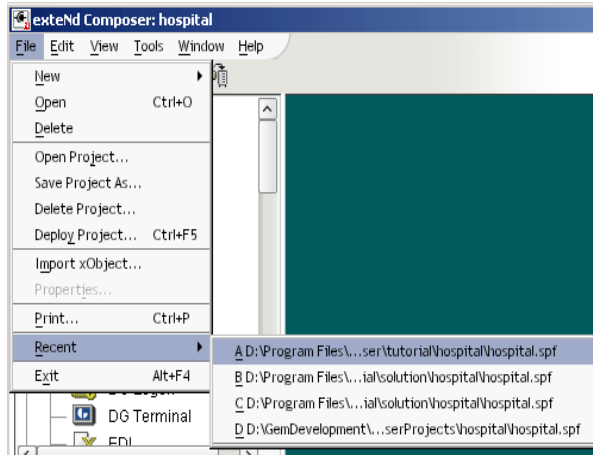


EXERCISE 2-1: Create an XML Map Component

- 1 If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.

When Integration Manager launches, it will open the last project you worked with in the application.

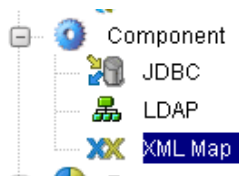
- 2 Check the title bar to insure you that the project listed is “hospital”. If not, open the “hospital” project by selecting **File>Recent** from the menu. You will see a list of the most recently accessed projects. Select “**hospital**” from the list.



- 3 Select the **Project Tab** in the Navigation Frame.

In this portion of the exercise you will create the map component that will be accessing the patient database. The input to this component will be the physician’s name. In the course of the exercise you will modify the Input DOM, and create the XML templates that model this input.

- 4 In the category pane highlight **XML Map** Component.



- 5 **RMB** (Right Mouse Button).

- 6 Select **New**.

The XML Map Component Wizard will appear.

- 7 Type **PatientLookup** in the Name text field.

- 8 Optionally, you may enter text in the Description fields, or leave them blank. The example below shows an entry in the remarks field.

Your wizard should look similar to the one depicted below.

Create a New XML Map Component

An XML Map component performs XML to XML mappings and content transformations. Its functionality is included in all Enterprise Connectors. Please enter a name and, optionally, a description for the XML Map Component. The name will appear in the Composer Detail Pane and in choice lists for XObjects in Composer. The name may not contain the characters: \/:? " < > . | Names are case insensitive (i.e. MyObjectName is the same as myobjectname).

Name:
PatientLookup

Description:
Purpose:
Input:
Output:
Remarks: retrieve patient information for the given doctor

Help ? < Back Next > Cancel

9 Click Next.

The Select XML Template dialog appears. You will be creating and saving an XML Template for the component later in the exercise. For this step, accept the default input and output templates provided by Integration Manager.

10 Click Next.

The Temp and Fault document dialog appears. You can think of Temp documents as a scratch pad area to use when manipulating data. Fault documents are used in fault handling. You will be using temp documents in a later lesson. The fault documents are beyond the scope of this introductory tutorial. For more information on them, please consult the *Novell Integration Manager User Guide*. For the purpose of this exercise, you will accept the default values provided by Integration Manager for these documents.

11 Click Finish.

The component editor window for the XML component will open.

Note that in the component editor you have an Input DOM and an Output DOM. The component name has been added to the Instance pane in the project tab, and the Action Model contains the XML Component Named “PatientLookup”.

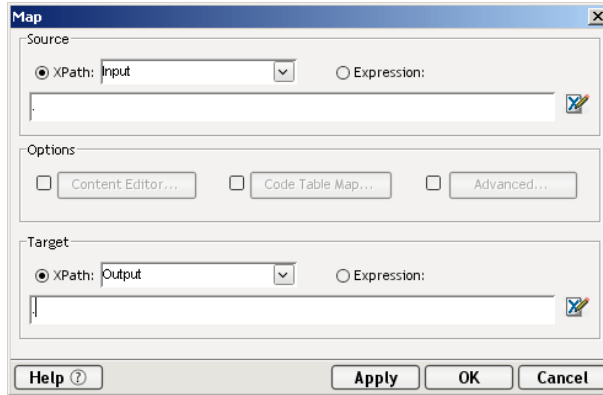
EXERCISE 2-2: Add a Map Action to the Component

1 Click to highlight “PatientLookup” in the action model.

2 RMB select **New Action>Map**, the map dialog appears.

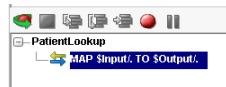
Note that the XPath radio button is selected for both the Source and the Target.

3 Type a, “.”, without the quotes in both the source and target text fields. The period is XPath notation indicating that the current node is to be copied.



4 Click OK.

A Map Action appears in the action model.

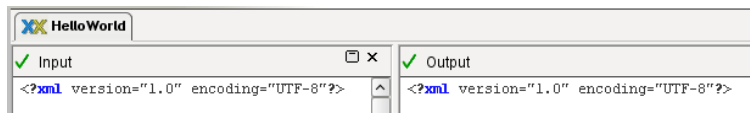


EXERCISE 2-3: Modify the Input DOM and Save as an XML Template

Integration Manager lets you view XML documents in three ways, tree, stylized, and text. The default view is tree. The easiest way to edit the XML document is to use the text view.

1 Select View>XML Documents>As Text from the menu.

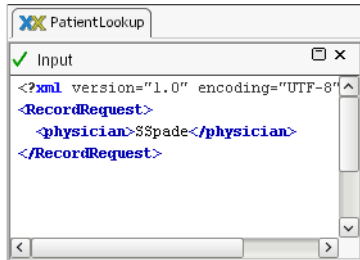
Note that the Input and Output DOM change to text view, which allows you to edit the structure of the XML document.



2 Click within the Input pane below the text “<?xml version="1.0" encoding="UTF-8"?>.”

3 Enter the following XML elements in the order they appear below, include the brackets and slashes. Integration Manager will generate an error if your XML document is not well-formed.

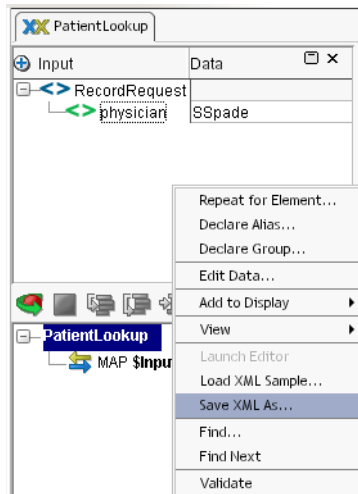
```
<RecordRequest>
  <physician>SSpade</physician>
</RecordRequest>
```



- 4 Select **View>XML Documents>As Tree**, the Input DOM displays physician as a child element of the root node RecordRequest.

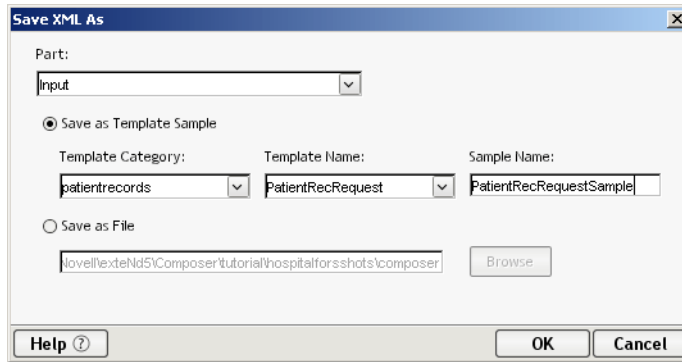
Next you will save the XML document you just created as a template to be used in later lessons. Any DOM that is visible in the component editor may be saved as an XML Template. A full discussion of XML templates is covered in Lesson 3

- 5 Click anywhere in the Input DOM, **RMB>Save XML As**.



The Save XML As dialog appears. Note that input appears in the dropdown list for the part. You will be saving your template as a template sample.

- 6 Select the **Save as Template Sample** radio button.
- 7 Type **patientrecords** in the Template Category dropdown box.
- 8 Type **PatientRecRequest** in the Template Name dropdown box.
- 9 Type **PatientRecRequestSample** in the Sample Name dropdown box.

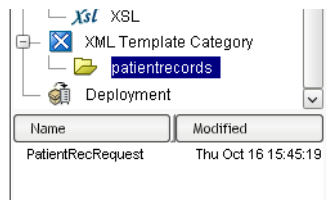


10 Click **OK**

Integration Manager creates a new folder under the XML Template Category named patientrecords.

11 Double click on the **patientrecords** folder in the Category pane.

Note that the XML template named PatientRecRequest is listed in the Instance pane. You will be using this template throughout the remainder of the tutorial.



EXERCISE 2-4: Animate the Map Component

1 Animate the map action by selecting **Animate>Start Animation** from the Integration Manager menu.

Note that the animation tool bar buttons become active above the action model.



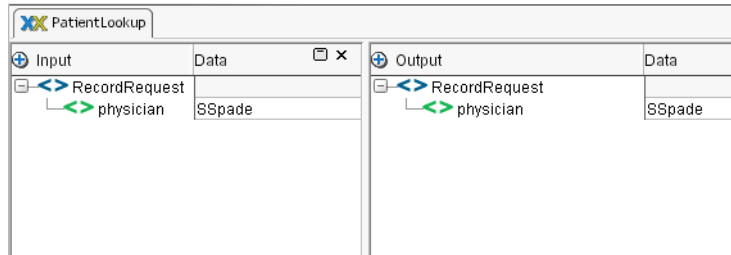
2 Hover the mouse over the buttons to determine their function.

3 Click the **Step Into** button twice to step through the component.

4 Click **OK** on the Animation dialog when you see the *Animation Completed* message.

5 Note that the physician element with the data “SSpade” has been mapped to the output DOM.

6 Select **File>Save** from the menu to save your work.



Summary of what you've done

You have accomplished the following tasks:

- ◆ Learned about Integration Manager Component Architecture
- ◆ Gained understanding about the Document Object Model (DOM)
- ◆ Created a XML Map Component.
- ◆ Added a XML Map Action to the component
- ◆ Created and saved an XML template
- ◆ Animated a Integration Manager component

Next lesson

In the next lesson you will learn about Schemas, XML Templates and XML Template Categories.

3

XML Templates

What you will learn

Lesson 3 introduces you to XML Templates and the XML files that comprise them. Templates provide frameworks on which to build components. You create XML templates that provide the input and output structure for the design, and test of components you build with Integration Manager.

You will learn about Integration Manager XML Templates:

- ◆ XML Template Categories
- ◆ XML Sample Documents
- ◆ XML Schema
- ◆ XSL Style Sheets
- ◆ XML Templates

What you will do

You will perform the following tasks using Integration Manager:

- 1** Work with XML Templates
- 2** Create an XML Template using the template wizard
- 3** Import an XML template from another Integration Manager project
- 4** Edit the PatientLookup XML Map Component to use the XML Templates

How long will it take? About 20 minutes

Integration Manager XML Templates

Integration Manager XML Templates are an organization of XML files in a functional group. They are used by the components and services you build. XML Templates model the elements and data a service or component will encounter. The type of documents that can be included in an XML template are XML Sample Documents, XML Schemas, and XSL Style Sheets

XML Template Categories

XML Categories provide a convenient mechanism for grouping like instances of templates. When you create a template category, it appears as a folder in the Category pane of the Integration Manager Navigation frame. When you create the XML template, you assign it to a category. The instances of the templates you have created appear in the Instance pane of the Navigation frame, when you select the appropriate XML Category.

XML Sample Documents

XML sample documents provide a representation of the runtime XML documents that a component or service will process. The elements and data in the sample document are the same as those encountered in practice. In B2B applications the sample documents are often supplied by your business partners. They may be used as inputs or outputs to a component or service.

XML Schema

XML Schema Definition (XSD) files are used to validate XML documents. They can be thought of as the contract that the XML document must fulfill to be interpreted properly by a service or component. Schemas are developed at design time between business partners. They act as an interface or description of the XML documents involved in the service. While XML documents are strictly concerned with data, schemas define the data's valid format and may also define data types. A schema can also declare namespaces to uniquely identify elements as belonging to a particular document vocabulary. Schemas allow extensibility, in that new custom data types may be defined within them.

The scope of schemas reach far beyond this tutorial. Entire books have been devoted to them. A good starting point to investigate them further is the [W3 Schools](#) website.

XSL Style Sheets

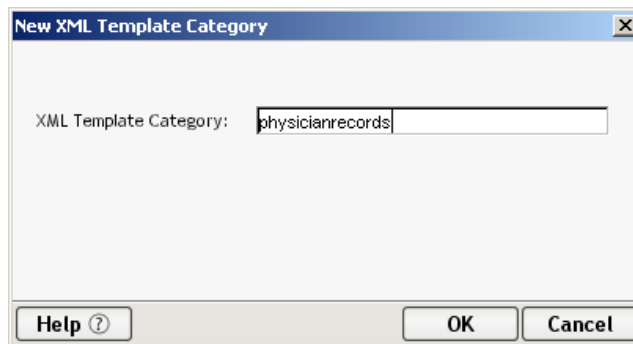
XSL Style Sheets may be included as part of the XML Template. While XML documents are strictly concerned with elements and data, XSL Style Sheets define how the data within the XML document is displayed. If your application is to be implemented as a page within a web browser, you may want to include a style sheet as part of your XML template.

XML Templates

XML Templates are a collected grouping of related documents that represent the information a component or service would receive at runtime. The primary use of XML Templates in Integration Manager is as a design and debugging aid. Components often have to handle a multitude of documents at runtime. These documents may have different optional elements that the components must handle gracefully. Templates allow you to test your component with the different document types your component may process at runtime.

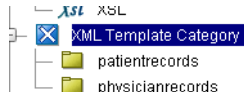
EXERCISE 3-1: Work with XML Templates

- 1 If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.
- 2 If you are not already in the “hospital” project, open it by either selecting it from the recent project list **File>Recent** on the menu. Alternatively, you may browse to the project location by selecting **File>Open Project**.
- 3 Select **XML Template Category** in the Navigation Frame.
- 4 **RMB>New**.
The New XML Category dialog appears.
- 5 Type **physicianrecords** in the dialog text field.



- 6 Click **OK**.

A new folder, “physicianrecords” appears in the Category pane of the Navigation frame.



- 7 Save your work by selecting **File>Save** from the menu.

EXERCISE 3-2: Create an XML Template using the template wizard

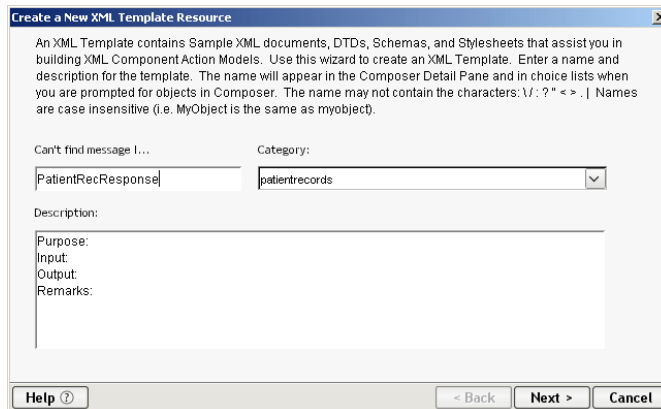
In this exercise you will create the XML templates that model the input and outputs of your service. The first template you create will model the final output of your service. It will combine the patient information retrieved from the database with the physician information extracted from the directory. In practice, a sample XML file might be supplied by the consumer of your service, you would then build your template using that sample file.

- 1 Click on the **patientrecords** folder.
- 2 **RMB>New** to launch the Create a New XML Template Resource Wizard.
- 3 Enter **PatientRecResponse** in the Name text field.

Notice that Integration Manager has entered the template category “patientrecords” in the category drop-down list.

- 4 (Optional) Enter text in the Description fields.

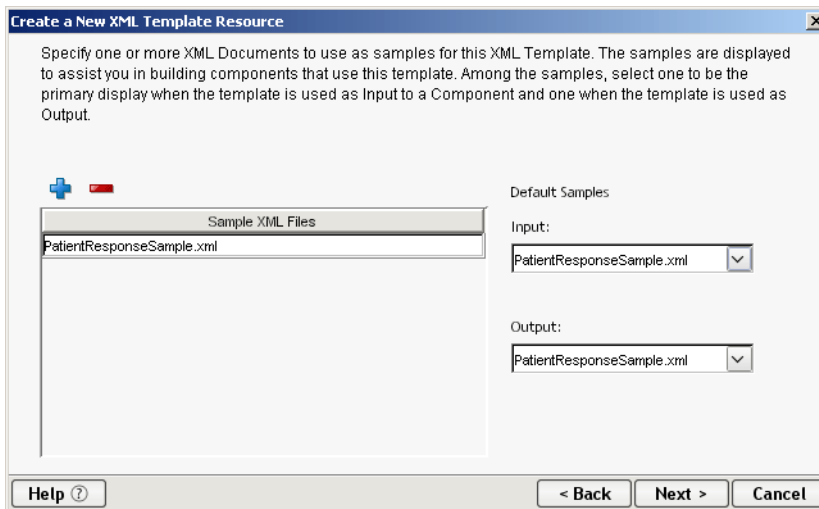
Your wizard should look similar to the one depicted below.



- 5 Click **Next**.
- 6 Add the XML Sample document to be used with the XML template.



- ◆ Click the **Add** button on the dialog.
- ◆ Navigate to the directory: `\tutorial\files`. This directory is located under the directory where Integration Manager was installed. Double click on the file: **PatientResponseSample.xml**.
- ◆ The wizard adds the file to the list and to the Default Samples drop-down lists for Input and Output.
- ◆ The panel should look like the one below.



- ◆ Click **Next**.

The sample document specified in the previous panel doesn't have an associated validation document. The wizard has selected the **None** radio button on this panel. Accept this default.

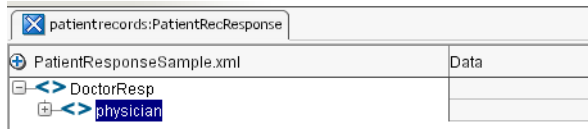
7 Click **Next**.

The Namespace and Prefix panel appears. Namespaces are used by Integration Manager Map Actions to resolve prefix usage between documents that belong to the same Namespace. For this exercise, you will leave this panel blank.

8 Click **Next**.

9 No stylesheet definition is associated with this template, click **Finish**.

The new template appears in the Navigation frame's Instance Pane and the XML Template Component Editor opens with the PatientRecRequest Template displayed. Note that the sample document's name appears in the title of the Part pane for the template.



Next you will create templates for the input and output of the physician directory.

- 1 Click on the “**physicianrecords**” folder.
- 2 **RMB>New** to launch the Import dialog.
- 3 Enter **PhysDirInq** in the name text field.
- 4 Click **Next**.
- 5 Add the XML Sample document to be used with the XML template.

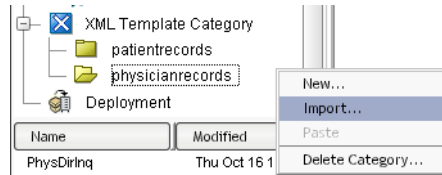


- ◆ Click the **Add** button on the dialog.
 - ◆ Navigate to the directory: **\tutorial\files**. This directory is located under the directory where Integration Manager was installed. Double click on the file: **PhysDirInqSpl.xml**.
 - ◆ The wizard adds the file to the list and to the Default Samples drop-down lists for Input and Output.
- 6 Click **Next**.
The sample document specified in the previous panel doesn't have an associated validation document. The wizard has selected the **None** radio button on this panel. Accept this default.
 - 7 Click **Next**.
The Namespace and Prefix panel appears. Namespaces are used by Integration Manager Map Actions to resolve prefix usage between documents that belong to the same Namespace. For this exercise you may leave this panel blank.
 - 8 Click **Next**.
No stylesheet definition is associated with this template.
 - 9 Click **Finish**.
 - 10 Open the template. Double click on the template name in the Instance pane. The XML Template Component Editor opens with the “PhysDirInq” template displayed.
 - 11 Save the XML Templates by selecting **File>Save All** from the menu.

EXERCISE 3-3: Import an XML template from another Integration Manager project

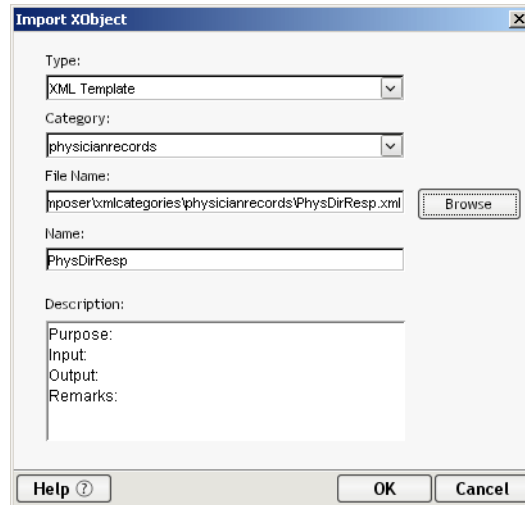
In this exercise you will learn how to import an XML template from another project. You will be importing the output template that will be used by the physician directory search component.


- 1 Right-click on the **physiciansrecords** folder under XML Template Category in the Category pane, then choose **Import**.



The Import XObject dialog appears.

- 2 Using the **Browse** button on the dialog navigate to the directory `\Composer\tutorial\solution\hospital\composer\xmlcategories\physicianrecords` under the location where you installed Integration Manager.
- 3 Double click on the file **PhysDirResp.xml**.
Your dialog should look similar to the one pictured here.

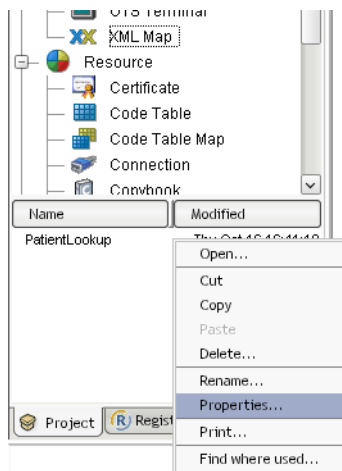


- 4 Click **OK**.
The template is added to the Instance pane.
- 5 Double click on the **PhysDirResp** template in the instance pane to open it.
- 6 Expand the root node in the template by clicking on the **plus sign** next to the root node.
- 7 Select **File>Save** from the menu, or use the **Save** button  on the Integration Manager toolbar to save your work.

EXERCISE 3-4: Edit the PatientLookup XML Map Component to use the XML Templates

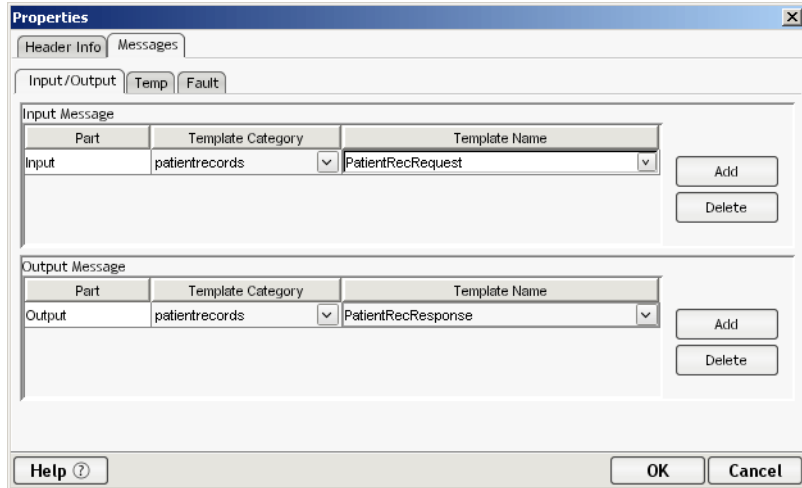
In this part of the exercise you will modify the PatientLookup map component to use the templates you have added to the project

- 1 Click on the **XML Map** component in the Category pane.
The PatientLookup map component appears in the instance pane.
- 2 **RMB** on the PatientLookup component in the instance pane.
- 3 Select **Properties** on the pop-up menu.

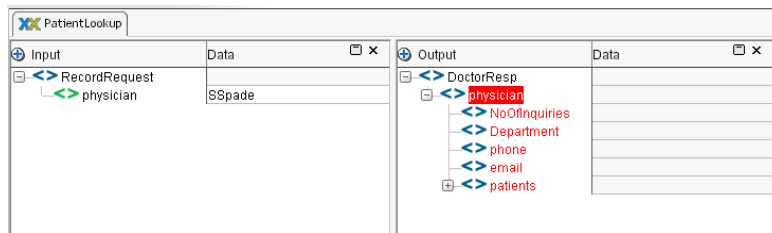


The properties dialog for the PatientLookup component appears.

- 4 Select the **Messages** tab on the Properties dialog.
The template category for the component is currently set to System and the template name to ANY.
- 5 Select the Template Category **patientrecords** from the dropdown list under the Input Message.
- 6 Select **PatientRecRequest** from the Template Name dropdown list for the Input.
- 7 Select the Template Category **patientrecords** from the dropdown list under the Output Message.
- 8 Select **PatientRecResponse** from the Template Name dropdown list for the Output.



- 9 Click **OK**.
- 10 Double click on the **PatientLookup** component in the Instance pane to open it. Note that input and output DOM reflect the templates you just added.



- 11 Save your work, select **File>Save All** from the menu.
- 12 Animate and step through the component using the animation tool bar.

Summary of what you've done

You have accomplished the following tasks:

- ◆ Learned about XML Templates and the documents that comprise them
- ◆ Created an XML Template Category
- ◆ Created an XML Template
- ◆ Imported an existing XML Template into a Integration Manager project
- ◆ Modified an XML Map Component that uses your templates

Next lesson

In the next lesson you will learn how to create a Integration Manager Service.

4

Web Services

What you will learn

This lesson will show you how to create a *web service* in the Integration Manager environment.

You will learn about:

- ◆ Integration Manager Service Basics
- ◆ Services vs. Components

NOTE: This lesson is just a very quick introduction to the basics of creating a new service in Integration Manager. It does not discuss WSDL or SOAP. For a detailed example of how to create and deploy a web service that uses WSDL and SOAP, see LESSON 8, “Publishing and Consuming Web Services”.

What you will do

You will perform the following tasks using Integration Manager:

- 1 Create a Web Service

How long will it take? About 5 minutes

Integration Manager Web Services

Web Service Definition

Web Services are applications that reside on a server, and execute when triggered by a client request. The request that initiates the service provides any input parameters the service needs to carry out its function. The service's response may be returned within the same transaction as the request, or it may occur as a separate operation.

Transactions in a web service often occur between business partners. One partner provides the service that the other consumes. For example; a catalog website may interact with an inventory supplier via a web service that, in turn, invokes the service of a third party shipping company.

Integration Manager web services accept XML documents as their input, use components and resources to act upon the data in the document, and return an XML document as their output.

Services vs. Components

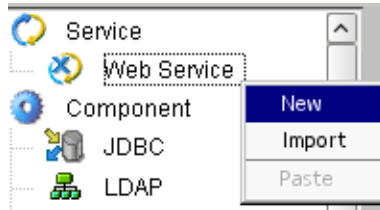
Services in Integration Manager are very similar to Integration Manager Components, they each have an Action Model, and can execute the same tasks. The wizard to create services is similar to the Component Wizard. Like components, services take XML documents as their input, and produce XML documents as their output.

The major difference between a service and a component is that the service is the unit of deployment. The service is the actual application that is invoked when the server receives a client request. Services are registered with the server, and may be published via a UDDI registry. Components reside within services, and perform units of work for the service. Services may also invoke other services to accomplish their tasks. When designing your application, keep in mind that the service is the highest level of your application. It presents the public interface to your application. The primary objective of the service is to coordinate the activities of the other services and components needed to fulfill the requirements of the service, and deliver the appropriate response to the client.

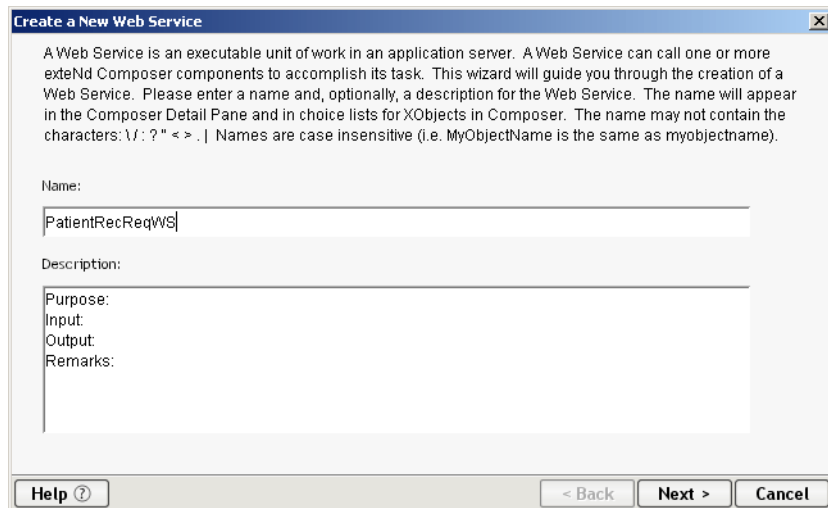
EXERCISE 4-1: Create a Web Service

The web service you will create in this exercise will be deployed in the next lesson. The service you will implement is intentionally very simple, your goal is to understand how to create the service. In later lessons, you will add more complex tasks to components and service.

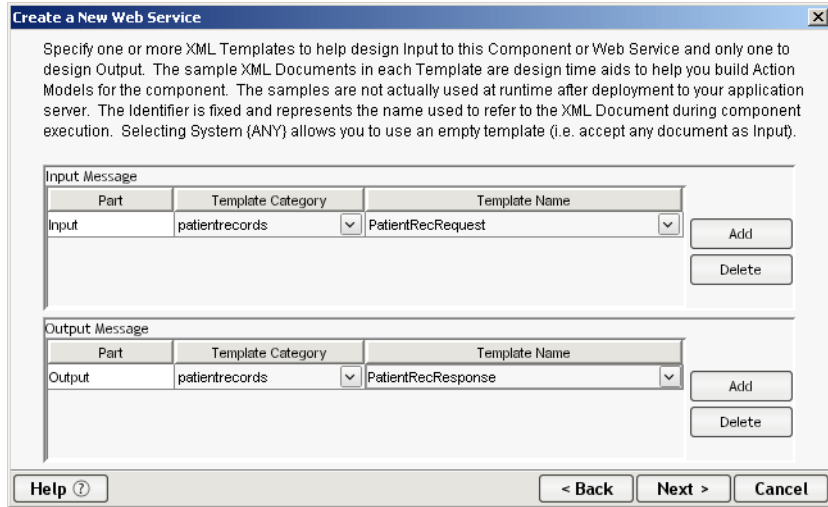
- 1 If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.
- 2 If you are not already in the “hospital” project, open it by either selecting it from the recent project list **File>Recent** on the menu, or browse to the project location by selecting **File>Open Project**.
- 3 Select the Web Service Category in the Navigation Frame.
- 4 **RMB>New**.



- 5 The Create a New Web Service Wizard appears.
- 6 Type **PatientRecReqWS** in the Name text field.
- 7 (Optional) Enter text in the Description fields.
Your wizard should look similar to the one depicted below.



- 8 Click **Next**.
- 9 Specify the XML Templates.
 - ◆ For the Input Message, select **patientrecords** from the Template Category dropdown list. Select **PatientRecRequest** as the Template Name.
 - ◆ For the Output Message, select **patientrecords** from the Template Category drop-down list. Select **PatientRecResponse** as the Template Name.



- ◆ Click **Next**.

10 You won't be using Temp or Fault documents in this section of the tutorial, so click **Next** to bypass this panel.

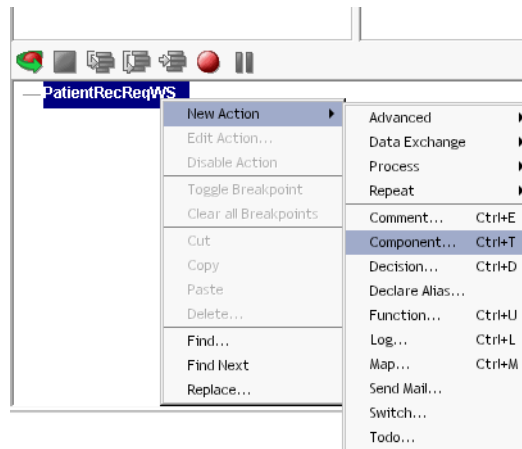
You will be deploying your service using SOAP as the service trigger in a later lesson. For this exercise, leave this panel blank.

11 Click **Finish**.

The component editor opens with the web service in the Native Environment pane. Next you will add an action to your web service to execute the PatientLookup XML component you created in Lesson 2.

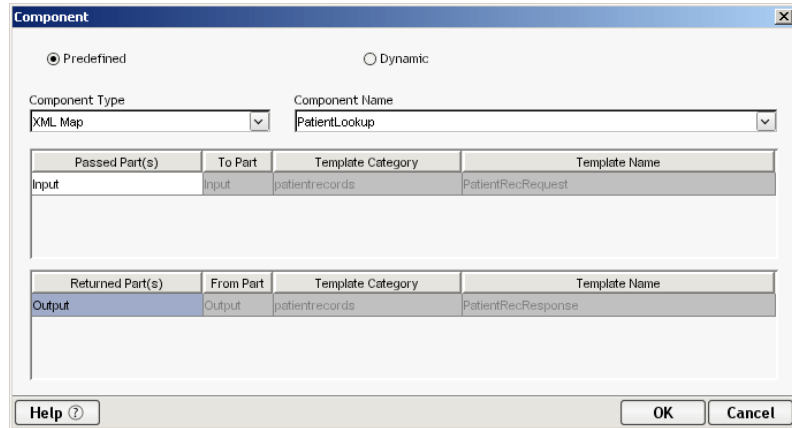
12 Highlight **PatientRecReqWS** in the action model.

13 **RMB>New Action>Component**.



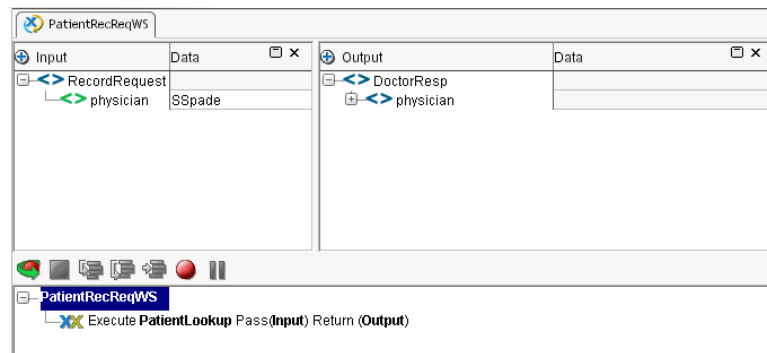
The Component dialog appears, with the Predefined radio button selected. At this point in the project, the PatientLookup component is the only component defined. The dialog defaults to this component. You will be passing the Input Part from the Web Service to the component and placing the output from the component into the Output Part of the service.

- 14** Select **Input** as the Passed Part, **patientrecords** as the Template Category, and **PatientRecRequest** as the Template Name.
- 15** Select **Output** as the Returned Part, **patientrecords** as the Template Category, and **PatientRecResponse** as the Template Name



- 16** Click **OK**.

The execute component statement is added to the action model. Your component should be similar to the one below.



- 17** Save your work, select **File>Save All** from the menu.
- 18** Animate and step through the service using the animation tool bar.

Summary of what you've done

You have accomplished the following tasks:

- ◆ Studied Web Services
- ◆ Discovered the difference between a service and a component
- ◆ Created a Web Service in Integration Manager

Next lesson

In the next lesson you will deploy your web service.

5

Deployment

What you will learn

Lesson 5 shows you how to deploy your Web Service.

You will learn about:

- ◆ Deployment Considerations
- ◆ Integration Manager Deployment Profiles
 - ◆ Server Profiles
 - ◆ Deployment Components
- ◆ Creating a Server Profile
- ◆ Service Triggers

What you will do

You will perform the following tasks using Integration Manager:

- 1** Create a Server Profile
- 2** Create a Deployment Object
- 3** Set up a Service Trigger for your Web Service.
- 4** Deploy your service

How long will it take? About 40 minutes

Integration Manager Web Service Deployment

When you have completed the design, building, testing, and debugging of your service on your local system, it's time to deploy the service to your application server.

Each application server supports its own method(s) of deployment. The following discussions pertain to the JBoss application server. If you are using another type of server, consult the documentation for your application server to learn how to deploy to your server.

Deployment Considerations

Before deploying your service, you need to take into account a number of factors ,as shown in the following table.

Factor	Details
Deployment Package	How will your service be packaged? Depending on your project requirements, you can choose to deploy your project from within the Integration Manager product or you can choose to create your own WAR or EAR file and deploy the archive manually.
Triggering	How will your service be instantiated? There are several options for triggering a service, including servlets, JSP, EJB, custom Java objects, scheduled tasks, e-mail, and (on Enterprise Edition only) SAP RFCs and/or JMS listeners.
Input Parameters	How will your service receive incoming data, and what will be the format and content of that data?
Resources	Will your service share resources with other services? How and where will you publish these resources?
Security	Will your service have restricted access? What about resource security?

In the tutorial you will package and deploy your web service using the Deployment xObject user interface of Integration Manager Enterprise Edition. You will first create a Server Profile (which is a prerequisite for deploying *any* kind of J2EE artifacts, using any edition of Integration Manager *or* Director), then you will create a Deployment xObject; and finally, you will deploy your project to the app server.

Server Profile

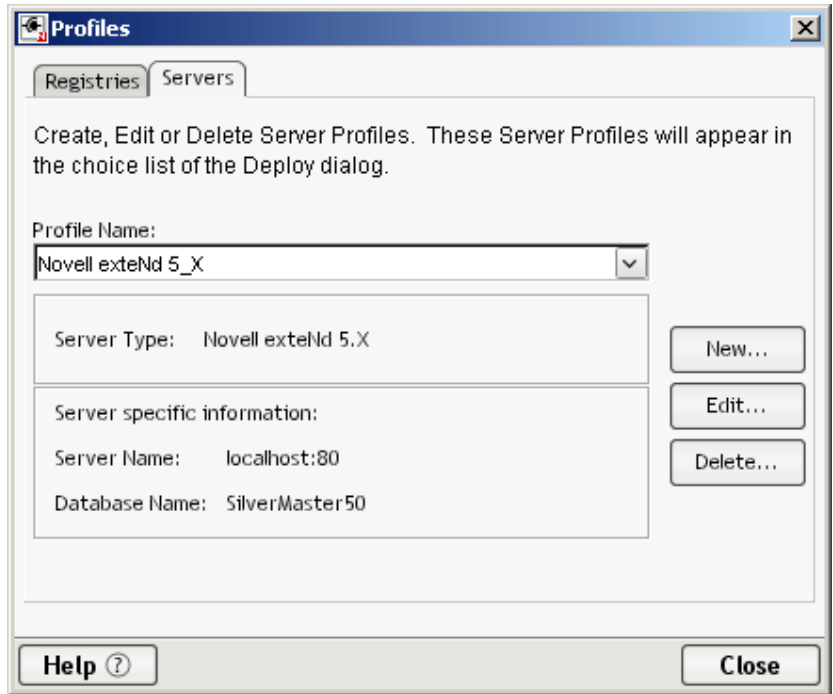
A Server Profile contains information characteristic to a given server, such as name (or IP address) and port number. Once a Server Profile is defined within Integration Manager, any project can be deployed to that server. If you've defined more than one Server Profile (for example, a local "development server" versus a QA or production server, etc.), you can choose the desired profile at deployment time, and deploy directly to the server of your choice without having to reenter or re-specify host and port parameters.

NOTE: Once you've created profiles for the server(s) you need to deploy to, there is no need to revisit the following procedure (unless you've upgraded servers or need to add an entirely new profile for some reason). This can be thought of as a onetime task.

EXERCISE 5-1: Creating a Server Profile

The first time you deploy using Integration Manager you need to define a server profile, as follows.

- 1** If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.
- 2** If you are not already in the "hospital" project, open it either by selecting it from the recent project list **File>Recent** on the menu, or browse to the project location by selecting **File>Open Project**.
- 3** Select **Tools>Profiles** from the Integration Manager menubar. The Profiles dialog appears.



- 4 Select the **Servers** tab if it is not already in front.
- 5 Click the **New...** button next to the Profile Name dropdown list. The New Server Profile dialog appears.

The screenshot shows a dialog box titled "New Server Profile". It has several input fields and a checkbox. The "Profile Name" field contains the text "tutorial". The "Server Type" dropdown menu is set to "JBoss 4.0.3". The "Server Name" field contains "localhost:8080". The "User Name" and "Password" fields are empty. At the bottom, there is a checkbox labeled "Use this server profile as the default for this project" which is unchecked. There are three buttons at the bottom: "Help" with a question mark icon, "OK", and "Cancel".

- 6 Type “tutorial” in the **Profile Name** text field.
- 7 Select the JBoss application server from the **Server Type** list.
- 8 Type the name and port number (separated by a colon) of your server in the **Server Name** text field. For example: `localhost:8080`
- 9 Enter your **User Name** and **Password** for accessing the server in question.
- 10 (Optional) You may elect to use this server profile as a default for all your Integration Manager Projects. To do so, select the checkbox at the bottom of the dialog.
- 11 Click **OK** to save the Server Profile.
- 12 Click **Close** on the Profiles dialog.

The server profile you just created will be available from *any* Integration Manager project.

Deployment xObject

The Deployment xObject is a special type of resource that wrappers deployment-configuration info specific to a particular collection of Integration Manager resources. The Deployment object specifies:

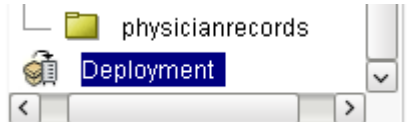
- ◆ The particular services to be deployed
- ◆ The types of *service triggers* that should be associated with the various services, along with parameters (URLs, Role names, etc.) specific to those triggers

- ◆ Any “publishable resources” that should be packaged into the deployment archive (such as WSDL, XSL, Image Resources, and other static resource types that should have archive-wide visibility)
- ◆ Any JARs that should be packaged into the deployment archive

It is possible to create more than one Deployment xObject within a given project. But for purposes of this tutorial, we will create just one Deployment xObject.

EXERCISE 5-2: Create a Deployment xObject.

- 1 Select the **Deployment** Category in the Navigation Frame.



- 2 **RMB>New.**

The Create a new Deployment xObject wizard launches.

Create a New Deployment Component

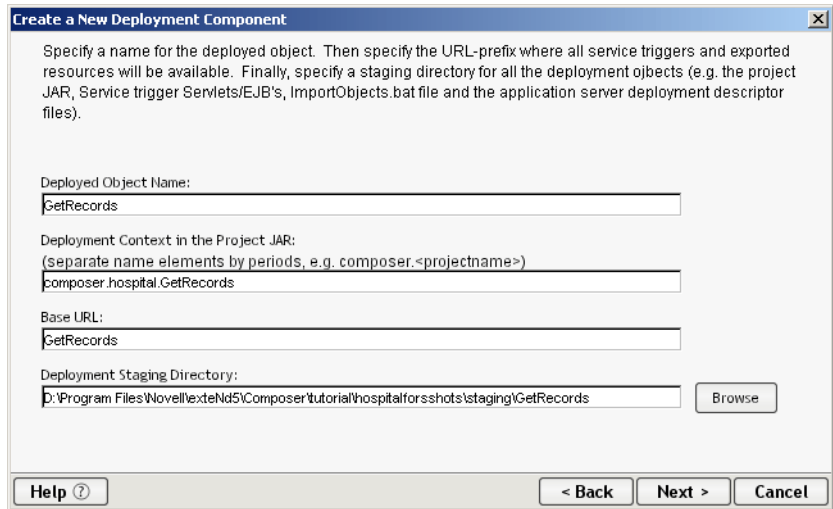
A Deployment object contains a complete and reusable deployment profile including the service triggers to create and the resources to deploy. This wizard will guide you through the creation of a Deployment object. Please enter a name and, optionally, a description of the Deployment object. The name will appear in the Composer Detail Pane and the choice lists of XObjects in Composer. The name may not contain the characters: \ / ? " < > . | Names are case sensitive.

Name:

Description:
 Purpose:
 Input:
 Output:
 Remarks:

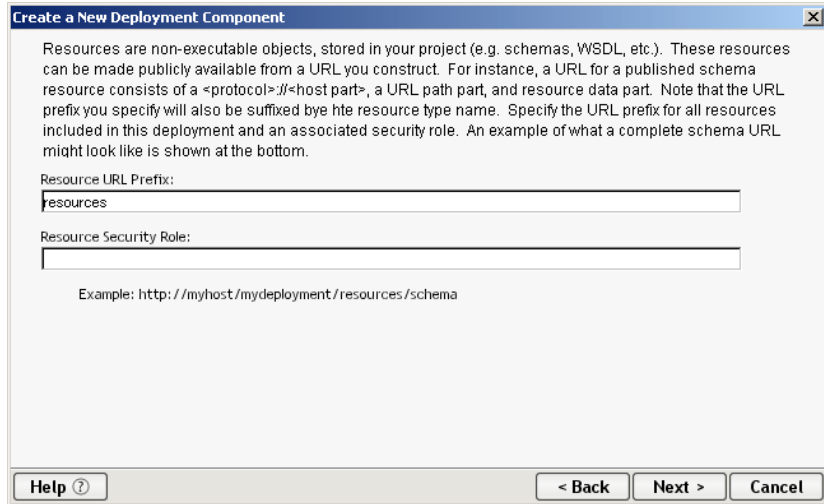
Help ? < Back Next > Cancel

- 3 Type **GetRecords** in the **Name** text field.
- 4 (Optional) Enter a description of the object.
- 5 Click **Next**. A new wizard panel appears.



- 6 You can accept the default values for Deployment Object Name, Deployment Context, Base URL, and Deployment Staging Directory.
 - ◆ The Deployment Object Name defaults to the project name you specified when you created your project.
 - ◆ The Context in the JAR is any arbitrary string of names separated by periods, such as, for example, `com.composer.tutorial`.

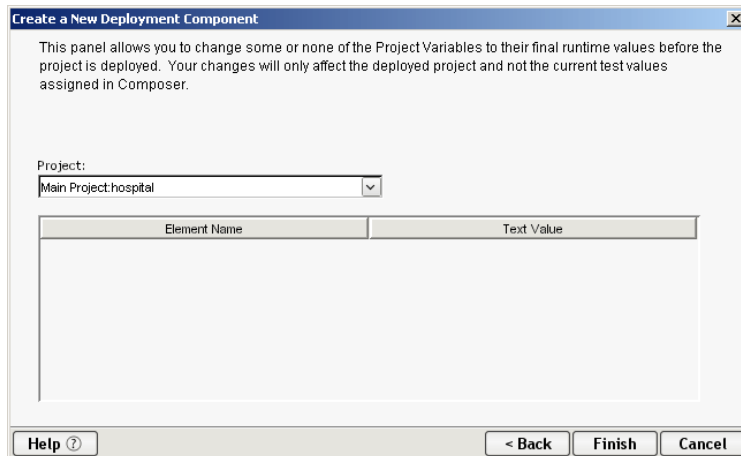
NOTE: As a best-practice, Novell recommends that you include the word “composer” in the deployment context of any applications built with Integration Manager, to distinguish them from apps built in Director or other environments.
 - ◆ The Base URL text field defaults to the Deployment xObject name.
 - ◆ The staging directory defaults to the one that was specified when your project was originally created.
- 7 Click **Next**. A new wizard panel appears.



Accept the default URL Prefix for the resources and leave the security role blank.

- ◆ The Resource URL Prefix defines where project resource files will be stored within the **EAR** file.
- ◆ The Security Role defines the access privileges for the project resources. For the purpose of the tutorial, you will leave the resource files fully accessible.

8 Click **Next**. A new panel appears.

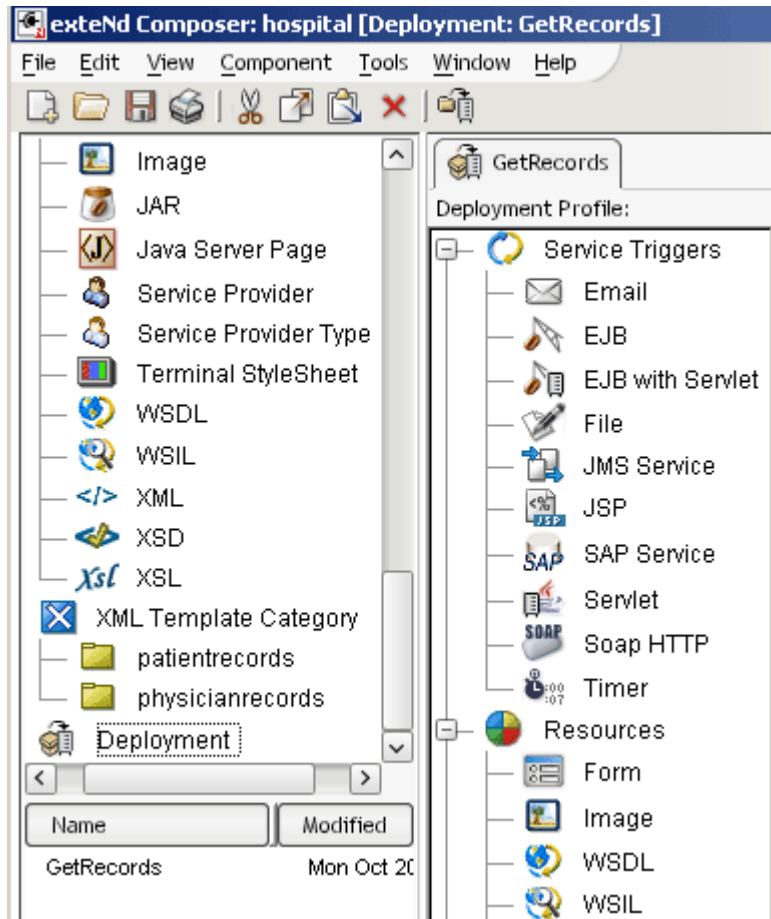


Your project's name appears in the Project dropdown list.

The purpose of this panel is to allow you to change the names or values of any Project Variables that you might want to change before deployment. (Project Variables might contain values that are relevant to a test environment but not to a production environment, or vice versa.) The tutorial project doesn't use any Project Variables, at this point, so it's safe to just ignore this panel for now.

- 9 Click **Finish** to complete creation of the Deployment xObject. You should see a new Deployment Object name show up in the instance pane of the explorer frame.

The Deployment xObject should appear in its own tree-view (to the right of Integration Manager's regular explorer frame) as shown in the screenshot below. If it does not, find the Deployment category in the regular explorer frame, and in the instance area, double-click the name of your deployment object.



- 10 To save your work, select **File>Save** from the menubar, or click the diskette icon in the main toolbar.

Service Triggers

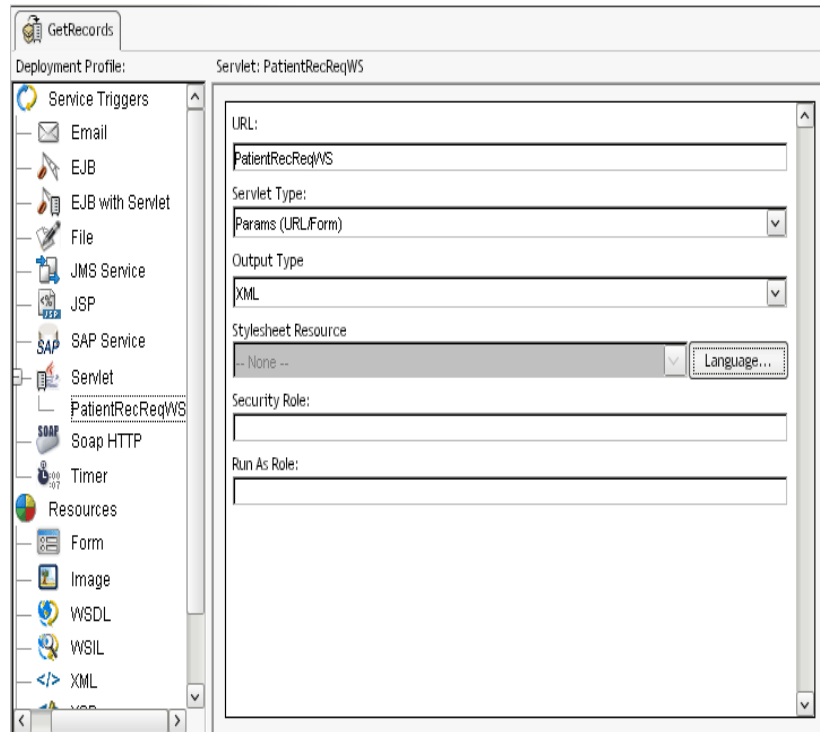
When planning the design of a web service, you should have a clear idea of the method by which the service will be triggered, and the parameters (or messages) that will need to be passed to the service. Integration Manager can create many types of service triggers for you (and associate them with URLs that you specify). You can also create your own custom service trigger classes, if you wish. An in-depth discussion of service triggers is outside the scope of this tutorial, but if you want to learn more, please refer to the *Novell Integration Manager User's Guide* in the chapter on deployment; and also see the separate *Integration Manager Enterprise Server User's Guide* for code-level discussions of trigger objects.

The following procedure leads you through the creation of JSP and servlet trigger types for the services contained in the tutorial project.

EXERCISE 5-3: Set up the Service Trigger(s) for your Web Service

The first method you will use to trigger your service is a servlet of the type Params(URL/Forms).

- 1** Expand the **Service Triggers** node in the **Deployment Profile** tree, if it is not already expanded.
- 2** Select **Web Service** at the top of Integration Manager's main explorer tree.
- 3** Select "**PatientRecReqWS**" from the instance pane.
- 4** Drag **PatientRecReqWS** to the Deployment Profile pane and drop it on the Servlet icon under Service Triggers.
"**PatientRecReqWS**" appears as a node under the Servlet branch, and the Servlet Properties sheet is displayed in the editor pane (what would normally be the Native Environment Pane). See below.

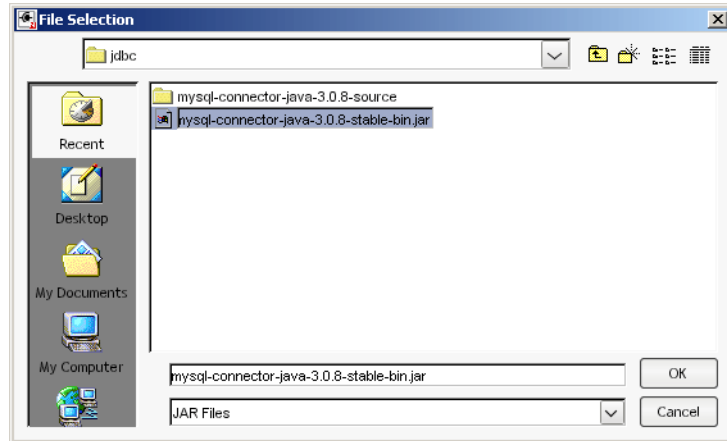


- 5 In the property-sheet pane, enter a **URL** string representing the final piece of the URL (after the final forward slash) that will trigger this servlet.
- 6 Select **Params (URL/Form)** for the Servlet Type in the dropdown list.
- 7 Select **XML** as the **Output Type** from the dropdown list.
- 8 There is no **Stylesheet Resource** associated with this web service, so skip this control.
- 9 Likewise, for purposes of the tutorial, you will leave the **Security Role** and **Run As Role** fields blank. These fields have the following effect:
 - ◆ The Security Role defines the access privileges for the service. (Roles are a J2EE 1.3 security construct and are implemented by the server; not by Integration Manager.)
 - ◆ The Run As Role defines the access privileges that PatientRecReqWS has to invoke other services.
- 10 Select **File>Save** from the menu.

EXERCISE 5-4: Import the MySQL JAR resource file.

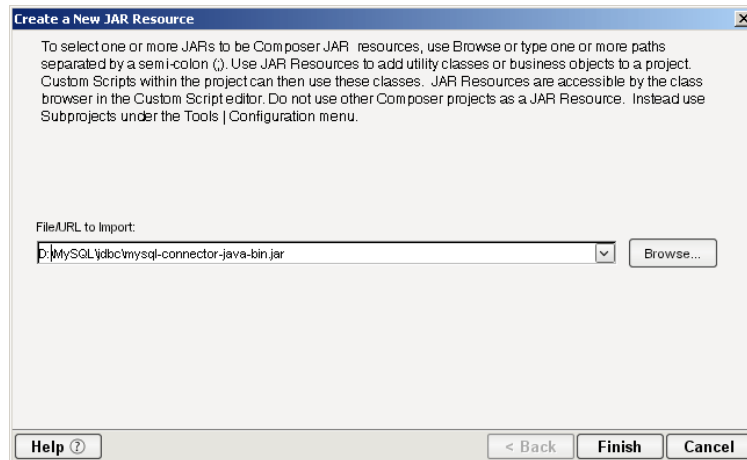
- 1 Right-click on **JAR** in the Resource section of the Category Pane, then select **NEW**.

- 2 Click the **Browse** button on the Create a New Jar Resource wizard panel.
- 3 Navigate to the `..\MySQL\jdbc` directory in the location in which you installed MySQL.
- 4 Double-click on the file JDBC JAR resource file (for example, `mysql-connector-java-bin.jar`).



- 5 Click **OK**.

The file name is added to the file/URL to import text window of the wizard panel.

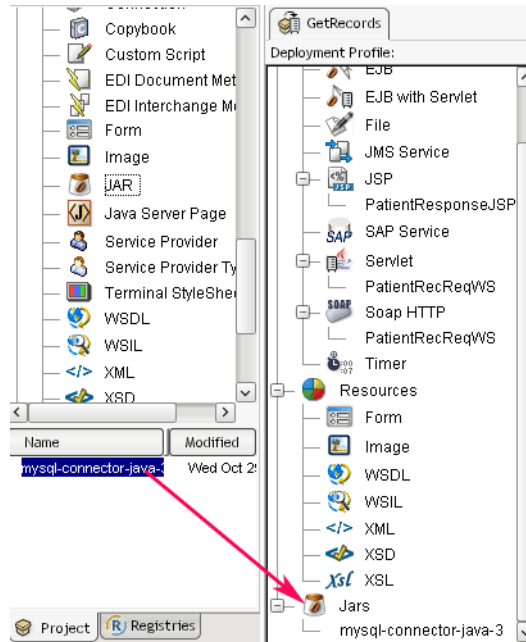


- 6 Click **Finish**.

The MySQL jar file is added to the Instance pane and opens in the component editor.

- 7 Double click the **GetRecords** tab in the component editor.

- 8 Drag the **mysql-connector-java-3** JAR resource from the Instance pane and drop it on the Jars element of the Resources tree in the Deployment Profile.



- 9 Select **File>Save** from the menu.

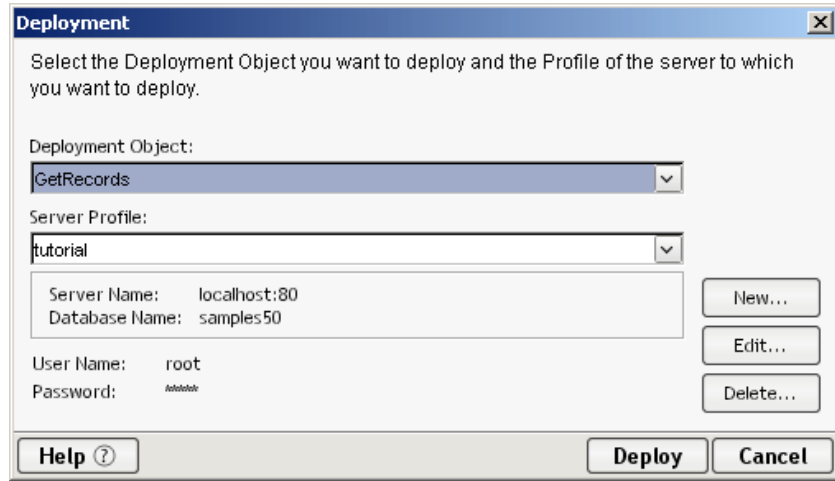
NOTE: The tutorial was written for deployment on the JBoss application server. If you wish to deploy to a different application server, please refer to the deployment specific documentation for that server type.

EXERCISE 5-5: Deploying to the JBoss application server

Before proceeding, make sure that you have set up a MySQL data source (see “Tutorial Setup” on page 8). Make sure that the JBoss application server is installed and running, with Integration Manager Enterprise Server also installed and running. (Integration Manager Enterprise Server is the runtime engine that executes and manages Integration Manager-built services in the application server environment.) These instructions assume the database and application server are installed on your local machine.

- 1 Start the MySQL server.
- 2 Start JBoss using the **start-jboss.bat** located in the /Novell/NIM6 directory.,
- 3 In Integration Manager, select **Deployment** in the Category Pane.
- 4 Select the **GetRecords** deployment component in the Instance pane.

- 5 From the Integration Manager menu, Select **File>Deploy**. The Deployment dialog appears.



The “GetRecords” deployment object is listed in the **Deployment Object** dropdown list.

- 6 Select your server profile from the **Profile Name** dropdown list.

- 7 Click **Deploy**.

Integration Manager will do a deployment validation to ensure that all required entries have been filled out in the properties sheet. If anything is missing, an entry will be made in the Todo list tab of the Message frame.

If your deployment is valid, your browser will launch. When the browser launches it will prompt you for the user name and password.

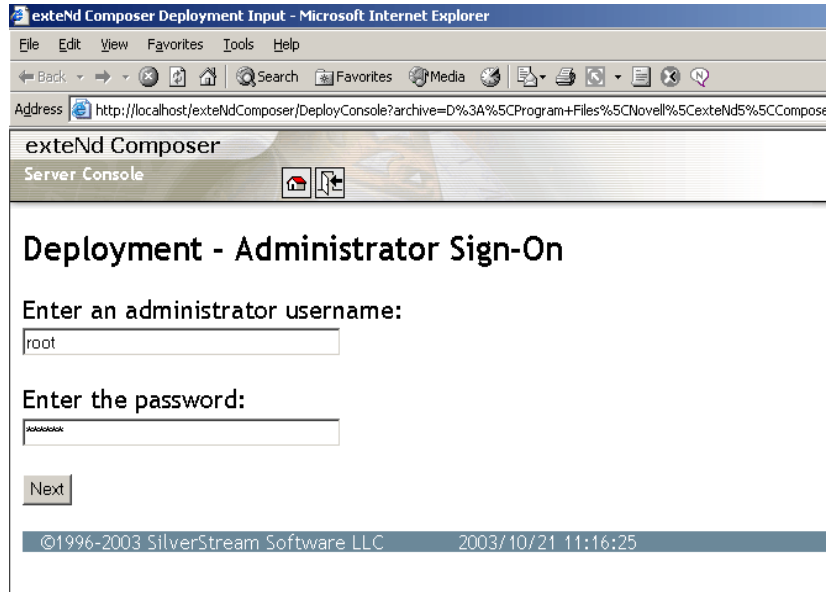
- 8 Enter the **user name** and **password** for the JBoss Composer Administrator (specified in the JBoss roles.properties and users.properties files). The default user name is “novell”, and the default password is also “novell”.

- 9 Click **OK**.

The browser opens and the administration sign-on page appears.

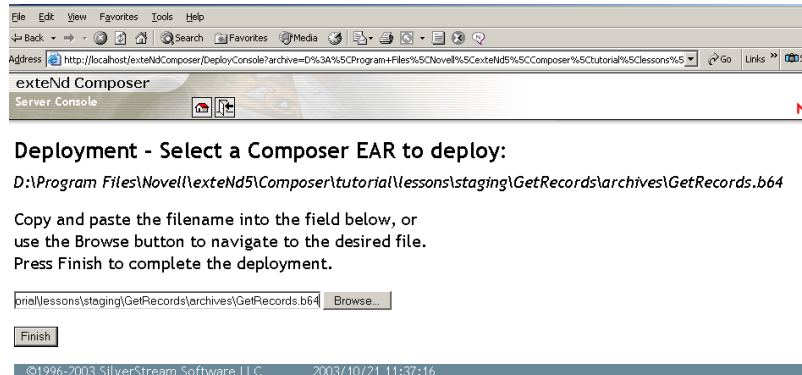
- 10 Enter your **user name** and **password** if the fields are not already populated for you.

- 11 Click the **Next** button.



12 Click **Next**.

13 Follow the directions on the browser page. (That is, cut and paste the fully qualified jar-file pathname that is displayed on screen into the text field. In Windows, *triple-click* the pathname to select all of it at once.)



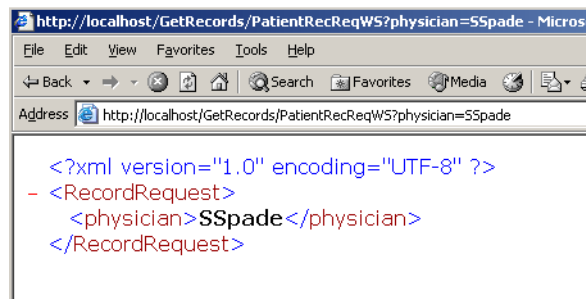
14 Click **Finish**.

15 When the deployment is complete the server will issue a deployment report.



- 16** Enter **http://localhost/GetRecords/PatientRecReqWS?physician=SSpade** in your browser address.

The service will run and display the XML file with SSpade in the physician element.



Summary of what you've done

You have accomplished the following tasks:

- ◆ Learned about deployment of Web Services in Integration Manager
- ◆ Created a Server Profile
- ◆ Created a Deployment xObject
- ◆ Set up a Service Trigger for your Web Service
- ◆ Deployed your Web Service

Next lesson

In the next lesson you will learn about Connections.

6

JDBC and LDAP

What you will learn

Lesson 6 introduces you to the JDBC and LDAP Connections, and their corresponding components.

You will learn how to create connection resources, and components that use those connections:

- ◆ Integration Manager Connection Resources and Connection Components
- ◆ JDBC Connection
- ◆ JDBC Component
- ◆ LDAP Connection
- ◆ LDAP Component

What you will do

You will perform the following exercises:

- ◆ Create a JDBC Connection
- ◆ Create a JDBC Component
- ◆ Access a Database using the JDBC Component
- ◆ Check for an active LDAP directory connection.
- ◆ Start the Novell exteNd LDAP Utility
- ◆ Create an LDAP Connection
- ◆ Create an LDAP Component
- ◆ Access a directory using the LDAP component

Integration Manager Connection Resources and Connection Components

Integration Manager uses connections to interface with databases, directories, and legacy data systems. The Connection Component contains information about drivers, time-outs, transport protocols, and any necessary parameters, specific to the establishment of a connection with the data source. Since Connections are specific to their data source, they are created for each data source that a Integration Manager project accesses.

Connection Components in Integration Manager allow legacy systems to become “XML aware.” That is, they provide a mechanism for the data in the system to be brought into a Integration Manager application as XML. This allows the Integration Manager application to operate on the data, and in turn, output it to another system. The reverse is also true. Integration Manager can take an XML input from a source, and update the legacy system via the Connection Component.

The tutorial will be using the JDBC and LDAP Connectors. You may have other, connectors in your Integration Manager installation, i.e. 3270. Each connector has its own user guide to help you understand its function.

JDBC Connection

JDBC is the JAVA-API for database connectivity. JDBC contains the classes and interfaces needed to establish database connections, execute SQL statements within a database, and process the result sets returned by the database.

The Integration Manager JDBC Connection provides connectivity to JDBC-accessible data stores. This connectivity is established utilizing a JDBC driver. A wide variety of database vendors supply JDBC drivers for their databases, enabling you to create connections to those databases within a Integration Manager application.

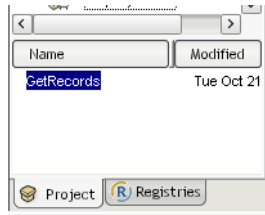
EXERCISE 6-1: Create a JDBC Connection

The tutorial uses the MySQL database that is packaged with Integration Manager, however Integration Manager supports any database that provides a JDBC driver.

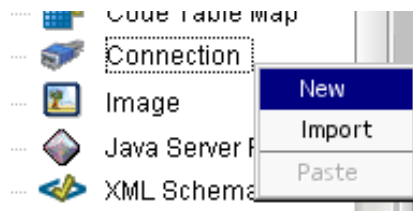
- 1 If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.

When Integration Manager launches, it will open the last project you worked with in the application.

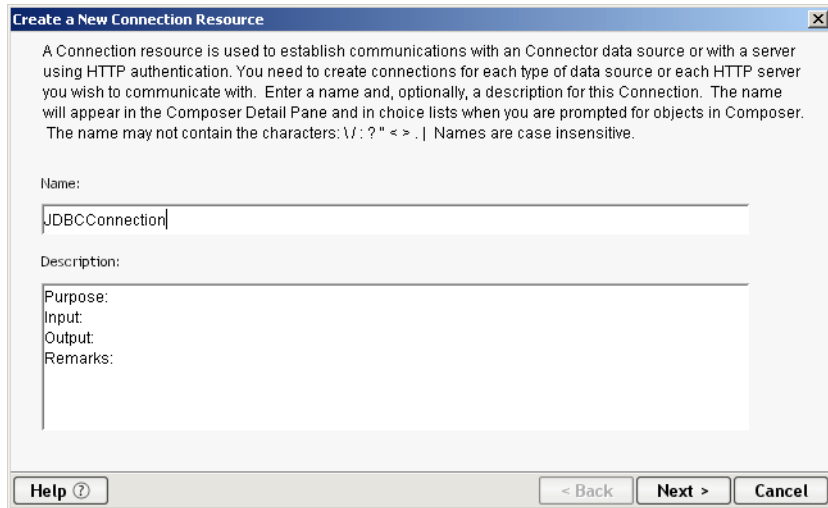
- 2 Check the title bar to ensure you that the project listed is “hospital”. If not, open the “hospital” project by selecting **File>Recent** from the menu. You will see a list of the most recently accessed projects. Select “hospital” from the list.
- 3 Select the Project Tab in the Navigation Frame.



- 4 In the Category pane, under Resource highlight **Connection**.
- 5 **RMB** (Right Mouse Button).



- 6 Select **New**.
- 7 The Create a New Connection Resource Wizard will appear.
- 8 Type **JDBCConnection** in the Name text field.
- 9 (Optional) Enter text in the Description fields.
Your wizard should look similar to the one depicted below.



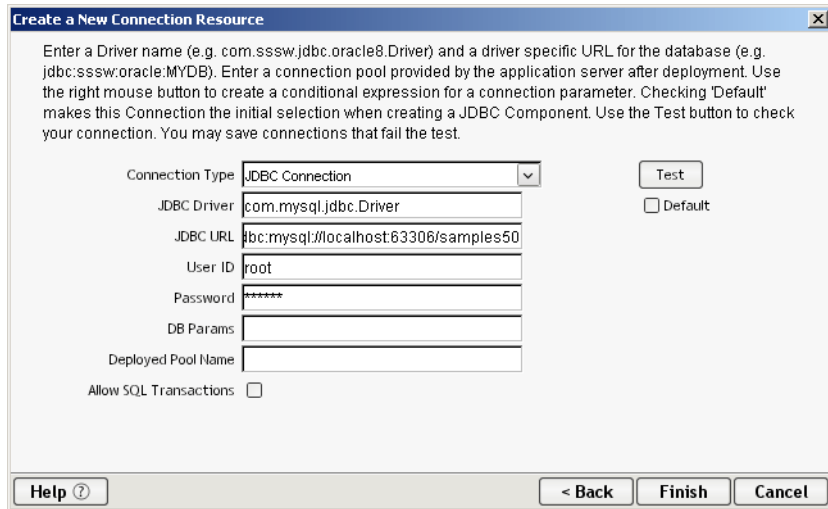
10 Click Next.

11 The Select Connection panel appears.

- ◆ Select **JDBC Connection** from the Connection Type dropdown list.

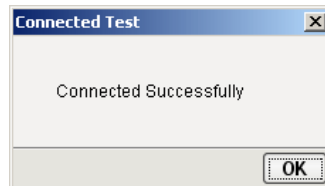


- ◆ Type **com.mysql.jdbc.Driver** into the JDBC Driver text field. This is the driver for the MySQL database.
- ◆ Type **jdbc:mysql://localhost:63306/samples50** into the JDBC URL field. The JDBC Driver text field contains the context where the database driver is packaged in your Integration Manager project. The JDBC URL is the location on your application server where the MySQL samples50 database can be found.
- ◆ Type your MySQL root user ID into the User ID field.
- ◆ Type the password for the user specified in the User ID field into the Password field. The password will be displayed as asterisk characters.
- ◆ Leave the remaining fields blank and the “Allow SQL Transactions” checkbox un-checked.



12 Click **Test**.

You should see a “**Connected Successfully**” dialog.



13 Click **OK**.

14 Click **Finish**, your connection is added to the Instance pane.

JDBC Component

A JDBC Component in Integration Manager uses a JDBC connection component to interact with a database. The component is similar to the XML Map Component, and can map, transform, and transfer data between XML Parts. The JDBC Component, can also create and execute SQL statements. The component interprets database result sets, converting them to XML. In addition, the component can make use of all available Integration Manager actions to operate on the data it receives.

EXERCISE 6-2: Create a JDBC Component

The JDBC Component you create will utilize the PatientRecReq XML template. The template contains the name of the physician, whose patients you want to retrieve from the database. Creating this component will fulfill the third requirement of your service; retrieve the patient data for the given physician.

- 1 In the Category pane under Components highlight **JDBC**.
- 2 **RMB>New**.
- 3 The Create a New JDBC Component Wizard appears.
- 4 Type **AccessPatientDB** in the Name text field.

Create a New JDBC Component

A JDBC component is used to push XML data into relational databases or pull data from them into XML documents. This wizard will guide you through the creation of a JDBC Component. Please enter a name and, optionally, a description for the JDBC component. The name will appear in the Composer Detail Pane and in choice lists when you are prompted for objects in Composer. The name may not contain the characters: \ : ? " < > . | Names are case insensitive (i.e. MyObjectName is the same as myobjectname).

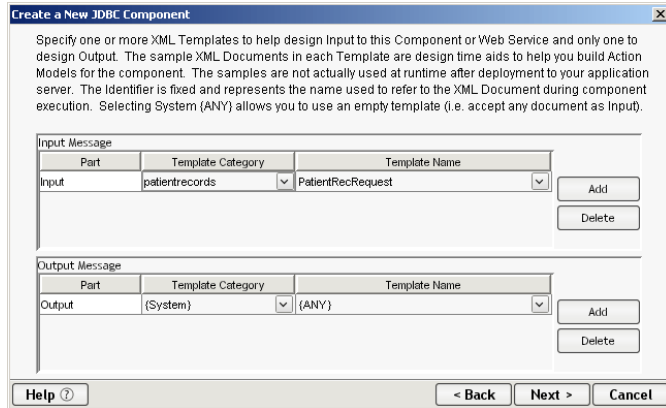
Name:
AccessPatientDB

Description:

Purpose:
Input:
Output:
Remarks:

Help ? < Back Next > Cancel

- 5 Click **Next**.
- 6 Select **patientrecords** from the dropdown menu for the Input Template Category.
- 7 Select **PatientRecRequest** as the Template Name for the Input Part.
Accept the defaults, System, for the Template Category, and Any, for the Template Name, for the Output Part.

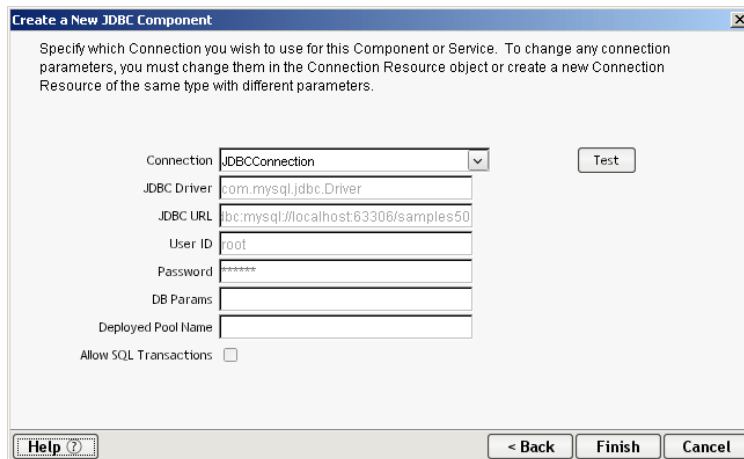


8 Click **Next**.

This section of the tutorial does not use Temp or Fault Messages.

9 Click **Next** to bypass this screen.

The wizard defaults to the JDBC Connection you created in the previous exercise.



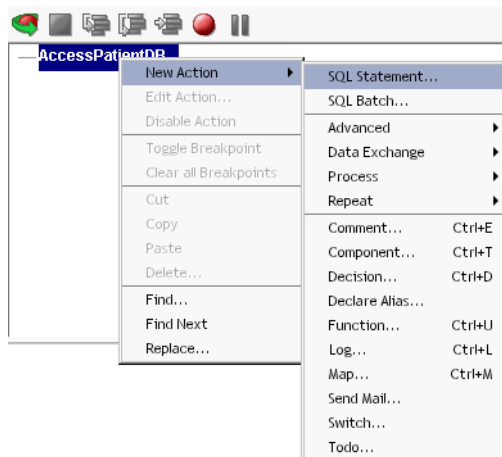
10 Click **Finish**.

The “AccessPatientDB” component is added to the Instance pane and the JDBC Component Editor opens.

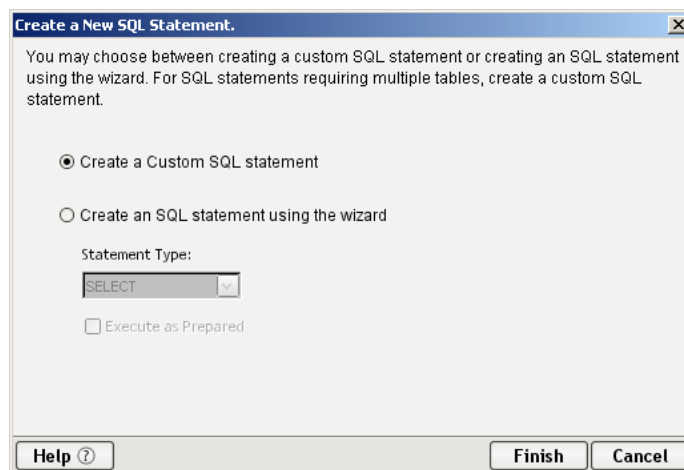
11 Select **File>Save**.

EXERCISE 6-3: Access a Database using the JDBC Component

- 1 Highlight the **AccessPatientDB** in the Action model.
- 2 Add a SQL Statement action to the component, **RMB>New Action>SQL Statement**.



- 3 The Create a New SQL Statement Wizard appears. The SQL Statement wizard aids you in building SQL expressions.

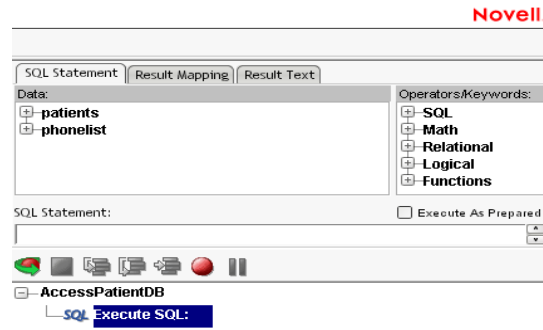


- 4 Select the **Create a Custom SQL** radio button.
- 5 Click **Finish**.

The JDBC Native Environment pane opens. Note the three tabs, SQL Statement, Result Mapping and Result Text. The SQL Statement editor is where you will build your SQL statement to query the database.

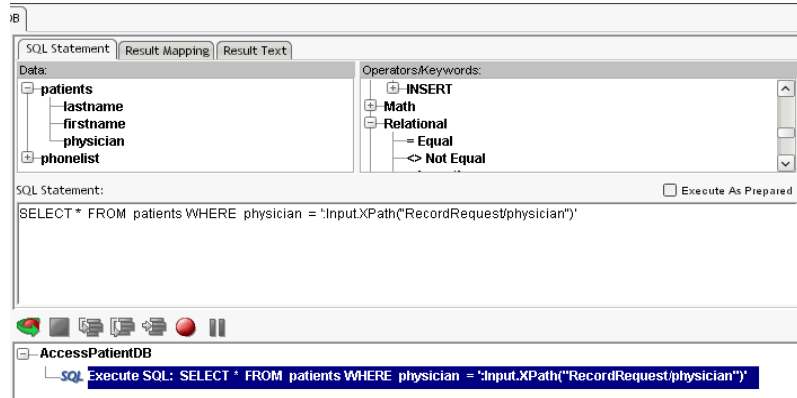
NOTE: If the editor is not visible hover the cursor over the border between the Native Environment pane and the Action Model. Use the double arrow to pull the editor window into view.

You will be adding a SQL Select statement that will search the database for patients whose physician is listed in the input document.



- 6 Click the **plus** sign before the **SQL** node in the SQL Operators and Keyword tree to expand the node.
- 7 Click the **plus** sign before the **Select** node in the SQL Operators and Keyword tree to expand the node.
- 8 Double-click **SELECT** then *****, then **FROM**, in the Operators and Keywords list tree. Note the phrase “**SELECT * FROM**” is added to the SQL Statement editor.
NOTE: You may have to use the scroll bar on the right side of the list to see all of the operators and keywords available.
- 9 Double-click the **patients** table in the Data tree, patients is added to the SQL statement.
- 10 Scroll down in the Operators and Keywords list and double-click on the **WHERE** operator to add it to the statement.
- 11 Expand the **patients** node in the Data list and double-click on the **physician** element.
- 12 Scroll down in the Operators and Keywords list to the **Relational** node.
- 13 Expand the node and double-click on the = **Equal** element.
Your statement should now read “**SELECT * FROM patients WHERE physician =**”.
- 14 Click in the **SQL Statement window** at the end of the statement to ensure the cursor is placed *after* the equal sign.
- 15 Highlight the **physician** element in the Input Part.

- 16** Drag the element from the Input physician node, and **Drop** it in the SQL Statement editor.

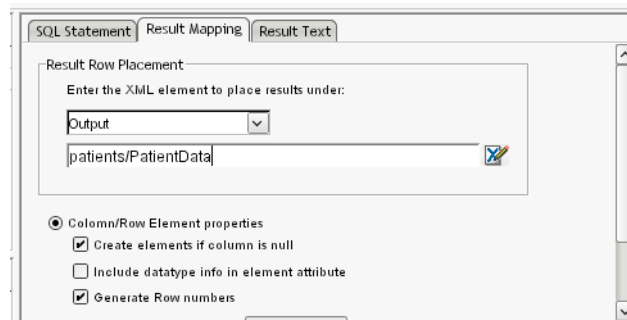


- 17** Your statement should now read: “**SELECT * FROM patients WHERE physician = ':Input.XPath("RecordRequest/physician")'**”.

NOTE: Integration Manager has created the XPath expression that accesses the data in the Input. When executed, this SQL statement will retrieve all the entries in the patients table whose physician element is equal to the data value in the physician element of the Input document.

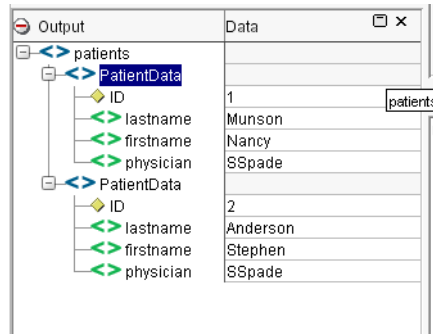
You can use the Result Mapping tab in the Native Environment pane to map the components output.

- 18** Select the **Result Mapping** tab in the Native Environment pane.
- 19** Delete the text **RESULTINFO/ROW** in the Result Row Placement textbox.
- 20** Enter **patients/PatientData** in the Result Row Placement textbox.
- 21** Using the scrollbar, scroll down in the pane to locate the **Generate Row numbers** checkbox.
- 22** Click the **Generate Row numbers** checkbox.

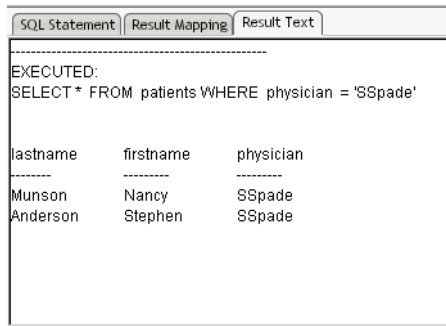


- 23** Select **File>Save** from the menu.
- 24** Execute the component, select **Component>Execute** from the menu.

25 Expand the **nodes** in the Output Part to examine the data.



The Result Text tab shows the exact text that was sent to the database, and the result set prior to being parsed into the Output Part.




Note that Integration Manager has placed the result set in the component Output part.

LDAP Connection

LDAP (Lightweight Directory Access Protocol) is a specification for accessing information within directories. It defines a standard protocol (much like HTTP) for communicating with *directory servers*.

The Integration Manager LDAP Connector allows you to access any directory that supports the LDAP protocol, and query the director using DSML (Directory Services Markup Language), which is a dialect of XML specific to directories. Integration Manager creates and manages DSML documents as it does any other type of XML document. You do not have to write DSML yourself: the LDAP Connector creates DSML-formatted queries for you, based on parameters you set using a point-and-click/drag-and-drop GUI.

 For more information on LDAP, please refer to the *Novell Integration Manager LDAP Connect User's Guide*.

EXERCISE 6-4: Check for an active LDAP directory connection.

A small LDAP directory tree and lightweight directory server (the Novell exteNd LDAP Utility) are included with Integration Manager. You will be using the test directory in the tutorial. The LDAP Utility must be running on your workstation in order to complete the remaining tutorial lessons.

If the LDAP Utility is running, the Windows OS tray will contain the JAVA icon as pictured below. (The Sent and Recv numbers will vary with your accesses to the directory.)



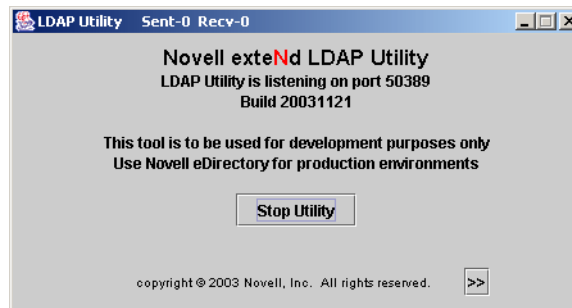
- 1 Double-click on the icon in the tray. A dialog with the title pictured below appears.



- 2 If the tool is **not running** on your work station, complete Exercise 6-5. *Otherwise, skip to Exercise 6-6.*

EXERCISE 6-5: Start the Novell exteNd LDAP Utility

- 1 From the Windows start menu select **Programs>Novell Integration Manager 6.0>Tools>LDAP Utility** to launch the Evaluation tool. A window appears, as shown below.

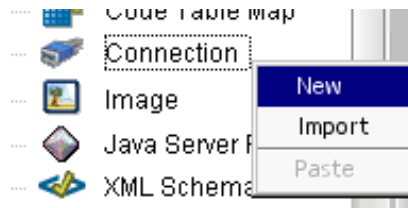


(If need be, you can locate the LDAP Utility in the `\tools` folder Integration Manager installation directory, and start it from there using the batch file provided.)

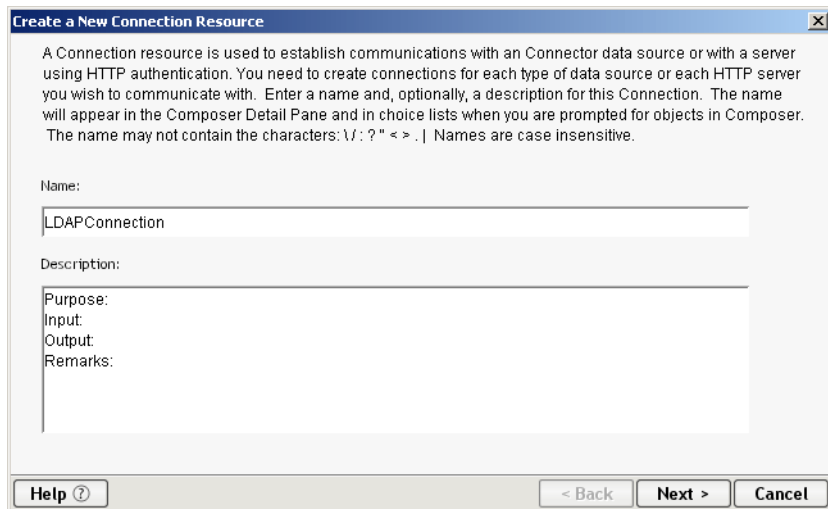
IMPORTANT: Be sure the LDAP Utility is listening on port 50389. You can see the current port number in the main window, above the Build Number. (If the port is not 50389, either modify the LDAP Utility's startup batch file and restart the server on that port, or modify your Connection Resource, below, as necessary to utilize the port that you want to use.)

EXERCISE 6-6: Create an LDAP Connection

- 1 In the Category Pane of the Navigation Frame select **Connection**.
- 2 RMB>New.



- 3 The Create a New Connection Resource Wizard appears.
- 4 Type **LDAPConnection** in the Name text field.

A screenshot of a dialog box titled 'Create a New Connection Resource'. The dialog contains a text area with instructions: 'A Connection resource is used to establish communications with an Connector data source or with a server using HTTP authentication. You need to create connections for each type of data source or each HTTP server you wish to communicate with. Enter a name and, optionally, a description for this Connection. The name will appear in the Composer Detail Pane and in choice lists when you are prompted for objects in Composer. The name may not contain the characters: \/: ? " < > . | Names are case insensitive.' Below the text are fields for 'Name:' (containing 'LDAPConnection'), 'Description:', 'Purpose:', 'Input:', 'Output:', and 'Remarks:'. At the bottom are buttons for 'Help', '< Back', 'Next >', and 'Cancel'.

- 5 Click **Next**.
- 6 Select **LDAP Connection** from the Connection Type dropdown list.
Integration Manager populates the dialog for you, setting the Host or IP Address to localhost, and 389 as the default Port Number.
- 7 Change the **Port Number** to **50389**.

- 8** You may leave the Base DN, User DN and Password fields blank. Leaving the Base DN blank will enable you to see the entire LDAP directory tree, from the root node within Integration Manager.
A blank UserDN and Password creates an anonymous bind when you connect to the directory.
- 9** Use the scroll bar to access the Time Limit field, enter **10000** in this field. (This is a millisecond value. It means that if no connection has been established after ten seconds, Integration Manager will stop trying to establish a connection.)
- 10** Leave the **Size Limit** value at its default of 1000. (This is strictly a design-time value that has to do with limiting the number of tree nodes Integration Manager will try to import for GUI/display purposes.)

The screenshot shows a dialog box titled "Create a New Connection Resource". It contains the following fields and controls:

- Connection Type: LDAP Connection (dropdown menu)
- Host or IP Address: localhost (text field)
- Port Number: 50389 (text field)
- Base DN: (empty text field)
- User DN: (empty text field)
- Password: (empty text field)
- Version: 3 (text field)
- Use TLS: (checkbox)
- Time Limit: (empty text field)
- Test button
- Default checkbox
- Help button
- Back, Finish, and Cancel buttons

- 11** Click the **Test** button to check the connection.
You should receive a “**Connected Successfully**” dialog. If not, check to be sure the port number matches that used by the LDAP Utility (see note further above, under Step 1); and be sure the utility is actually running.
- 12** Click **OK**.
- 13** Click **Finish**.
The LDAP Connection Resource will appear in the Instance pane in the Navigation Frame.

LDAP Component

The LDAP Component provides communication, via the LDAP Connection Resource, with an LDAP directory server. It supports the full set of Integration Manager actions, allowing you to transform, map and manipulate directory data. In addition, the LDAP Component can create and execute DSML statements. You will typically use Temp documents to store transient DSML query and response info.

As with XML Map Components, you can use XML Templates to define the structure and mapping of the LDAP Component's Input and Output message parts.

The LDAP Component you create in this exercise will query a directory in order to obtain a given physician's *title*, *e-mail address*, *phone number*, and *hit count*. The hit count indicates the number of directory inquiries that have been made for that particular physician.

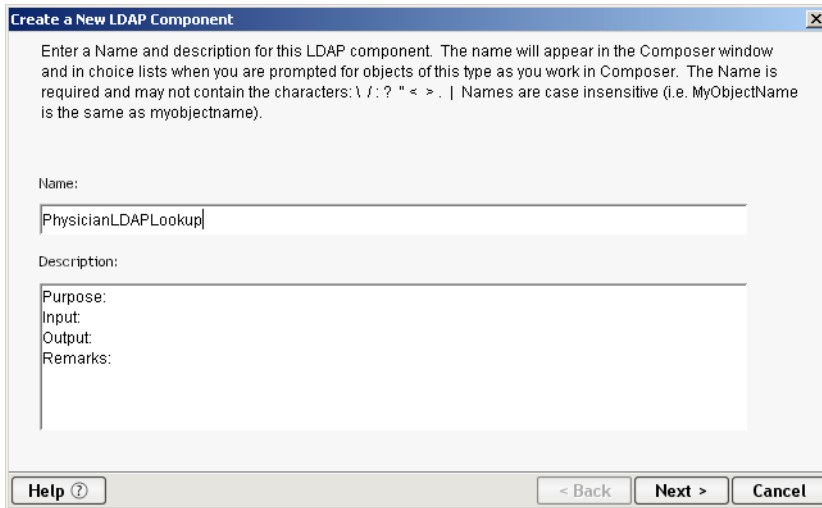
EXERCISE 6-7: Create an LDAP Component

- 1 In the Category pane under Components, highlight **LDAP**.
- 2 RMB>New.

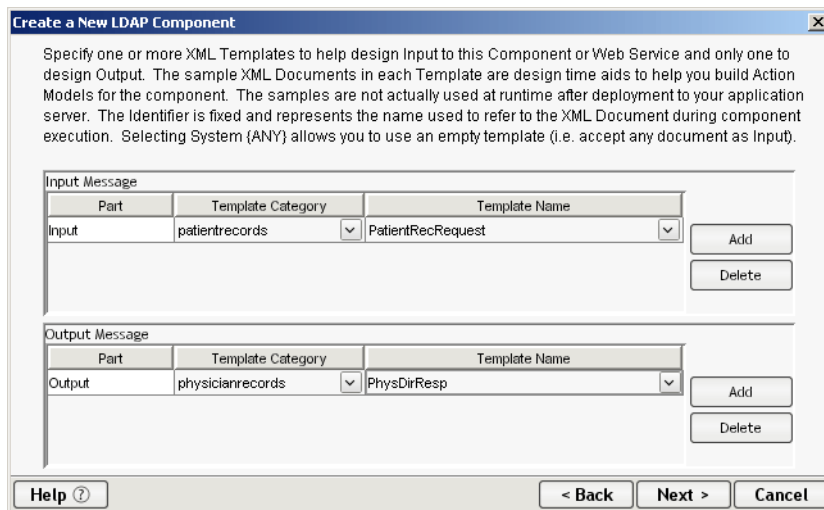


The Create a New LDAP Component Wizard appears.

- 3 Type **PhysicianLDAPLookup** in the Name text field.



- 4 Click Next.
- 5 Select **patientrecords** as the Template Category for the Input part from the dropdown list.
- 6 Select **PatientRecRequest** as the Template Name for the Input part from the dropdown list.
- 7 Select **physicianrecords** as the Template Category for the Output part from the dropdown list.
- 8 Select **PhysDirResp** as the Template Name for the Output part from the dropdown list.



- 9 Click Next.

- 10** Your LDAP component will be using temp docs for creating DSML requests and responses.

Click the **Add** button twice in the Temp Message portion of the panel.

Two Temp parts are added to the panel, *Temp* and *Temp1*. You may leave the Template Category and Template Name fields at their default values.

You won't be using Fault docs in this section of the tutorial.

The screenshot shows a dialog box titled "Create a New LDAP Component". It contains two sections: "Temp Message" and "Fault Message".

Temp Message Section:

Part	Template Category	Template Name	
Temp	{System}	{ANY}	Add Delete
Temp1	{System}	{ANY}	

Fault Message Section:

Part	Template Category	Template Name	
_SystemFault	{System}	{Fault}	Add Delete

At the bottom of the dialog are buttons for "Help", "< Back", "Next >", and "Cancel".

- 11** Click **Next**.

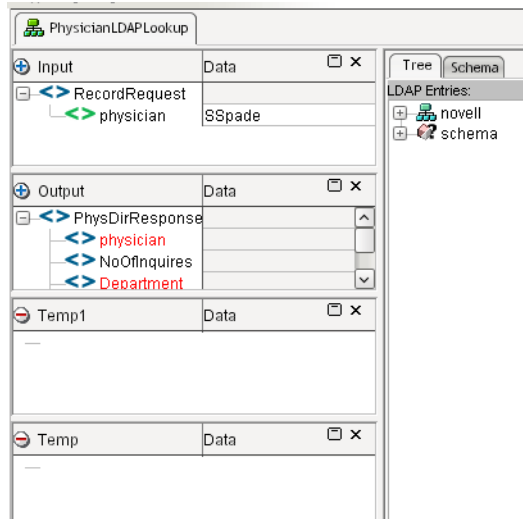
The wizard defaults to the LDAP Connection you created in the previous exercise.

The screenshot shows the "Create a New LDAP Component" dialog box at the "Connection" step. It contains the following fields and controls:

- Connection: LDAPConnection (dropdown)
- Host or IP Address: localhost
- Port Number: 60389
- Base DN: (empty)
- User DN: (empty)
- Password: (empty)
- Version: ?
- Use TLS:
- Time Limit: 10000
- Test button
- Help, < Back, Finish, Cancel buttons at the bottom.

- 12** Click **Finish**.

The LDAP Component is added to the Instance pane of the Navigation Frame, and Integration Manager launches the LDAP Component Editor. Note that there are 4 DOM in the Component, Input, Output, Temp and Temp1.

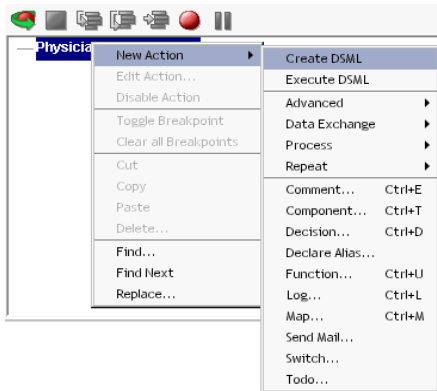


Access a directory using the LDAP component

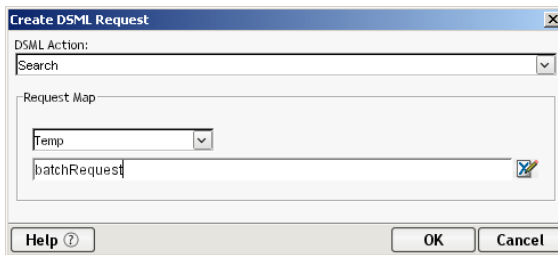
In this exercise, you will query the LDAP directory. The search will return information from the directory for the physician in the Input DOM. After completing the search, you will either create an initial hit count for this physician, or increment the existing hit count.

EXERCISE 6-8: Add a Create DSML action to the Action Model

- 1 Highlight the “PhysicianLDAPLookup” component in the Action Model.
- 2 RMB>New Action> Create DSML.



The Create DSML Request dialog appears. The dialog defaults to the Search action.

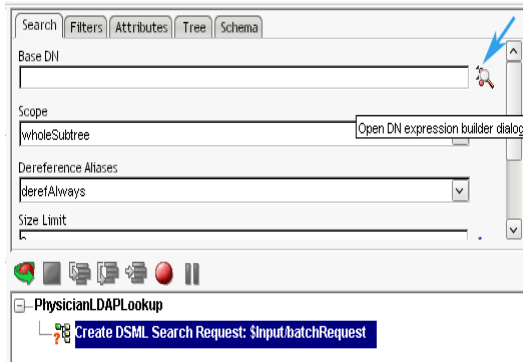


- 3** Select **Temp** for the Request Map.
- 4** Click **OK**.

A Create DSML Search Request action is added to the Action Model.

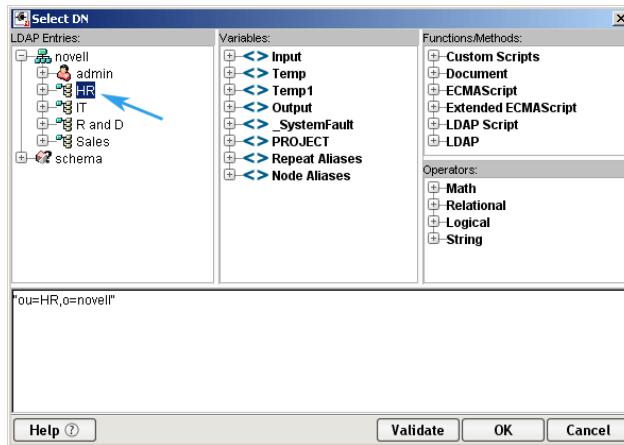
EXERCISE 6-9: Set the Depth of the Directory Search

- 1** In the Native Environment Pane, with the Search Tab forward, click the **DN** button next to the Base DN text field.



The Select DN expression builder opens.

- 2 Expand the **Novell** node in the LDAP Entries Pane, click on the **plus** sign next to the node to expand it.
- 3 Double-click on the **HR** node, the expression "ou=HR,o=novell" is added to the dialog's expression window.



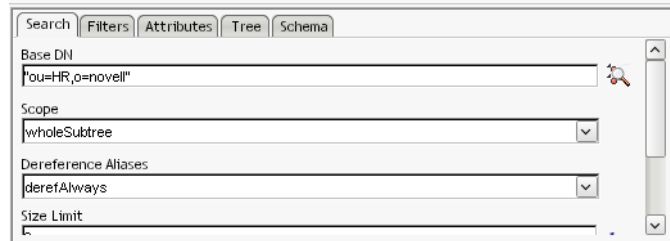
- 4 Click **OK**.

The Base DN field of the Search Tab in the Native Environment pane should now contain the expression you just built surrounded by quotes. This expression indicates that the search of the directory should begin at the "HR" organization unit, under the organization "novell".

- 5 Select **wholeSubTree** from the dropdown list for Scope on the Search Tab.

The scope of the search will be the base node, and entire subtree under the Base DN, the HR node and all the nodes under it.

- 6 Select **derefAlways** from the dropdown list for the Dereference Aliases field. The search will dereference aliases (directory entries that contain a pointer to another entry in the directory) and search both. Leave the remaining settings at the default values.

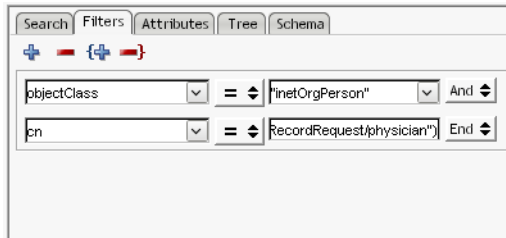


EXERCISE 6-10: Create the Filter for the Search

The filters tab helps you build expressions that determine what data is returned by the search. Only the data from directory entries matching the filter specification will be returned.

- 1 Select the **Filters** tab in the LDAP Native Environment pane. Accept the default of objectClass in the first dropdown list on the tab.
- 2 Select “**inetOrgPerson**” from the second dropdown list positioned after the equal sign on the tab.
- 3 Click the left most **Plus** button on the dialog to add an additional row to the filter. Note that the relational operator at the end of the first expression changes from End to And.
- 4 Select “**cn**” from the first dropdown list of the new row. Next you will use drag and drop to create the XPath expression representing the data within the input DOM physician element.
- 5 Highlight the **physician** node in the Input DOM.
- 6 Using the mouse drag the **physician** element from the Input DOM and drop it on the second dropdown list of the row you added in the filter tab. The expression **Input.XPath("RecordRequest/physician")** is added to the dropdown.

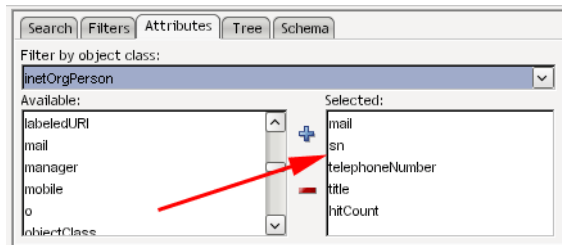
The filter you created will return directory items where the objectClass is equal to “inetOrgPerson” and whose “cn” (common name) is equal to the physician element of the input document.



EXERCISE 6-11: Select the attributes to be returned by the search

The attributes tab lets you select what fields you would like returned from the directory for the searched object. You will be retrieving the physician's *surname*, *e-mail address*, *hitCount*, *telephone number* and *title* from the directory.

- 1 Select the **Attributes** tab in the Native Environment Pane
- 2 Select **inetOrgPerson** from the Filter by object class dropdown list.
- 3 While holding the control key (Ctrl) down, highlight the following items from the Available scroll list on the tab: **mail**, **hitCount**, **sn**, **telephoneNumber**, and **title**.
- 4 Click the **Plus** sign between the two windows to copy the objects to the Selected window.



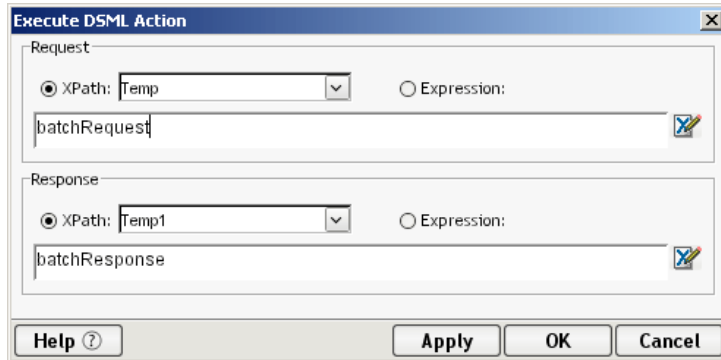
- 5 Select **File>Save** from the menu to save your work.
- DSML expressions must be created and executed in two steps, next you will add an execute DSML action to the Action Model

EXERCISE 6-12: Add the Execute DSML action to the Action Model

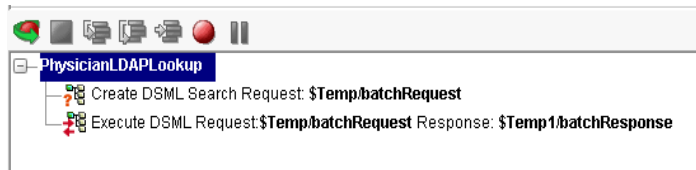
- 1 Highlight the "Create DSML action".
- 2 **RMB>New Action>Execute DSML**.
The Execute DSML Action Dialog appears.
- 3 Select **Temp** as the Request part.
- 4 Select **Temp1** as the Response part.

5 Click **OK**.

Leave the remaining values at the default settings supplied by Integration Manager.



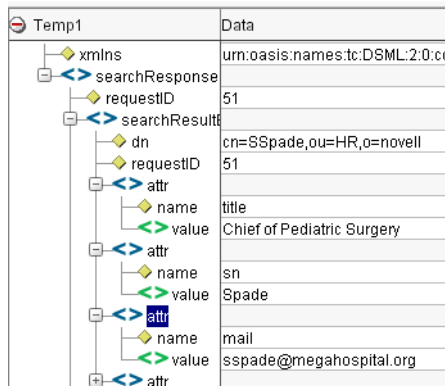
Your action model should look similar to the one portrayed:



6 Select **Component>Execute** from the menu to test your LDAP Component.

7 Expand the **searchResponse** node in the Temp1 DOM and explore the information.

You should see data similar to the following screenshot in your Temp1 Part.



8 Select **File>Save** from the menu to save your work.

Summary of what you've done

You have accomplished the following tasks:

- ◆ Learned about Integration Manager Connections and Components
- ◆ Become familiar with the JDBC Connect and the JDBC Component
- ◆ Learned about the LDAP Connect and LDAP Component
- ◆ Created a JDBC Connection
- ◆ Created a JDBC Component
- ◆ Accessed a database using the JDBC Component
- ◆ Created an LDAP Connection
- ◆ Created an LDAP Component
- ◆ Accessed a directory using the LDAP Component

Next lesson

In the next lesson you will add new functionality to the LDAP Component you just built, including a *modify* operation that actually writes to the directory. You'll also see how to map data into and out of DSML queries, check for error conditions, and perform conditional branching.

7

Basic Integration Manager Actions

What you will learn

In Lesson 7 you will become familiar with some of the basic actions available in Integration Manager that will help you accomplish the requirements of your projects. You add the actions to your components and web services that act on the data and elements passed to it in the Input Part, transforming them to the desired Output Part. If you are familiar with programming languages, you will find Actions analogous to the programming constructs found in any modern computer programming language. However, you don't need to be a programmer to use Integration Manager. The most important items to understand when adding actions to your Integration Manager components are the requirements of the component. The tutorial instructs you in some of the most common actions used when creating Integration Manager projects. For information on the remaining actions supported by Integration Manager, please refer to the *Novell Integration Manager User Guide* and system help.

You will learn about the Integration Manager Action Model and Actions:

- ◆ Comment Action
- ◆ Function
- ◆ XPath and ECMAScript (The Integration Manager Expression Builder)

What you will do

You will perform the following tasks using Integration Manager:

- 1 Edit the PhysicianLDAPLookup Component
 - ◆ Add a comment to the Action Model.
 - ◆ Add a Decision Actions

- ◆ Add Map Actions
 - ◆ Add Function Actions
 - ◆ Add a Log Action
- 2** Edit the PatientLookup XML Map Component
- ◆ Add Component Actions
 - ◆ Add Map Actions

How long will it take? About 30 minutes

Integration Manager Action Model and Actions

Components and Web Services in Integration Manager both use action models to achieve their goals.

In Integration Manager, an Action Model is the set of instructions that a component implements to complete a task. The inputs to the action model are one or more XML Input Parts. The Input Parts may be data from XML documents, or from non-XML sources via Integration Manager Connections, or both. The Component operates on the Input Parts by applying the actions in its Action Model to the data and elements of the Input Parts. The result of the Action Model is the Output Part of the component.

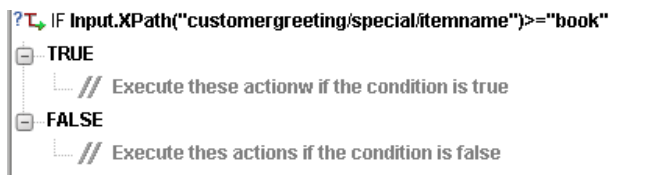
The architecture of Integration Manager allows you to create components that break your project into logical units of work. Each Component performs a specific task needed by your web service or another component. You might have a LDAP Component that reads in data from a directory, another JDBC component that accesses a database, and a third Map Component that combines the output of those components into an XML Output Part, based on a common element in each.

Comment Action

Comment Actions are used to document your Action Models. Comments aid you in remembering what your action model does and are a valuable resource in the maintenance of your projects. Since comments are not executed, they do not add to run-time overhead. You may add comment actions anywhere in the Action Model.

Decision

The Decision statement is one of the conditional actions supported by Integration Manager. It provides you with the ability to choose a course of action based on a provisional statement. When you add a Decision Action to Integration Manager, you specify a condition on which to base the branch in the Action Model. One branch of actions is executed if the condition evaluates to “True”, the other if the evaluation is “False”. Decision actions allow your components to be flexible, and react to different run-time conditions in an appropriate manner.



Log Action

Log actions are designed to provide customized reporting capabilities (design-time as well as runtime) for Integration Manager applications. You can exercise fine control over the degree of reporting desired, by the use of Log Level settings.

Examples of places where Log actions are useful include, writing error information to a file or console, debugging web services and components, capturing cycle specific information in looping constructs, and to track actions, i.e. hits on a URL.

Component

The Component action calls and executes another component or web service in your Integration Manager project. You use Component Actions to process a unit of work in your project. When you call a component, you pass in four parameters, the Component Type, the Component Name, the Input Parts you want the component to process, and the name of the Output Part where you want the component’s output placed when it returns to the calling component. You can use Pre-defined or Dynamic parameters to call the component.

Pre-defined parameters are defined at design time, when you build the Action Model for the calling component. They are static and remain the same each time the component executes.

Dynamic parameters are generated at run-time while the calling component is executing. They vary dependent on the circumstances at the time of the call.

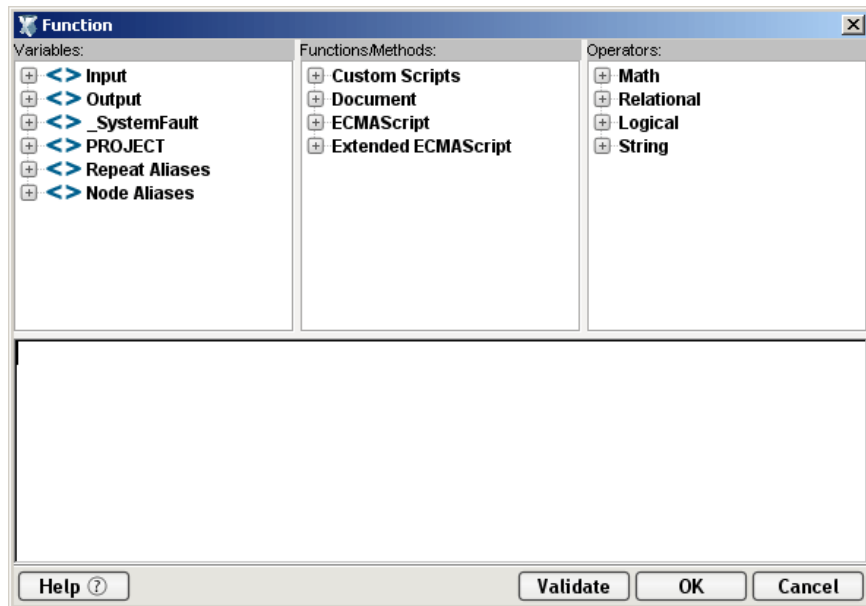
Function

A function action in Integration Manager executes a task for a component, within the frame of the component or service. They differ from Component actions, in that a function is not an entity in your project, although it may make use of parameters and produce an output. You can use a functions to operate on individual or multiple Part Elements, perform a complex operations i.e. a mathematical operation, or access a file system or URL.

Function actions make “function calls”, that is, they call either ECMAScript or previously created Custom Script functions that will execute the desired operation. In addition, you may register Java methods in the Custom Script Resources that you can invoke with a function action. Custom Scripts are outside the scope of a beginning tutorial, refer to the *Novel Integration Manager Users Guide* for more information on this topic.

XPath and ECMAScript (The Integration Manager Expression Builder)

The Expression Builder allows you to use either XPath or ECMAScript expressions to access and operate on XML document elements.



XPath is the primary means of addressing elements in an XML document. XPath uses pattern matching to locate a specific element or node in a document. XPath also provides simple expression statements that allow you to manipulate element data. Integration Manager provides pick lists in the Expression Builder to generate XPath expressions. You select the elements from the pick list, and the expression is built for you by the editor. You will find the complete XPath Specification at <http://www.w3.org/TR/xpath>.

The second method of addressing Part elements in Integration Manager is via ECMAScript. ECMAScript (ECMA-262 and ISO/IEC 16262) is a scripting language that you can use to do more complex manipulation of objects within Integration Manager. ECMAScript allows you to extend the operations you can perform in the Integration Manager environment. ECMAScript is a powerful tool, to find out more about it, consult the *Novell Integration Manager User Guide*.

Edit the PhysicianLDAPLookup Component

In exercises 7-1 to 7-5 you will modify the **PhysicianLDAPLookup** component you created in lesson 6. You will use a decision action, to determine if your directory search was successful. Upon a successful search, you will return the physician attributes retrieved from the directory via the Output part. If the directory search fails, you will log the failure to System Out.

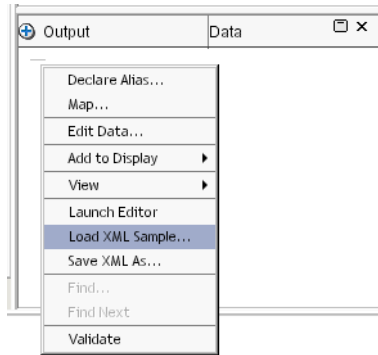
You will also be incrementing the *hitCount* for the physician that is the object of your directory search. If the *hitCount* element does not exist, you will create the element by modifying the directory.

Make sure the application server, the Novell exteNd LDAP Utility and the MySQL service are all running on your system before continuing. (See Lesson 6 for detailed instructions.)

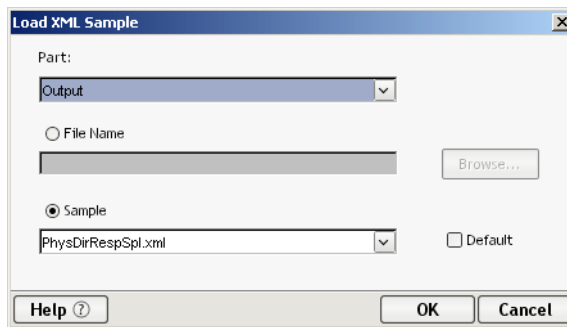
EXERCISE 7-1: Set up the component for edits

- 1** If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.
- 2** If you are not already in the “**hospital**” project, open it by either selecting it from the recent project list **File>Recent** on the menu, or browse to the project location by selecting **File>Open Project**
- 3** Select the **LDAP** Component in the Category pane.
- 4** Double-click on the **PhysicianLDAPLookup** component in the Instance pane. You will first need to execute the existing component to obtain data from the directory to use in when adding actions to the component.
- 5** Execute the component by selecting **Component>Execute** from the menu. You will need to reload the output XML template to complete the map actions
- 6** Click in the **Output** Part window.

7 RMB>Load XML Sample.



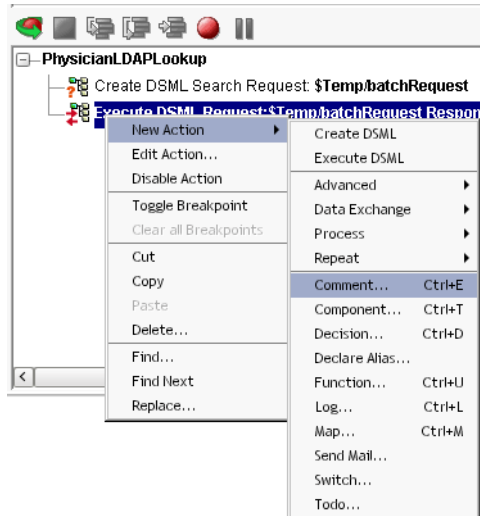
8 Click OK on the Load XML Sample dialog.



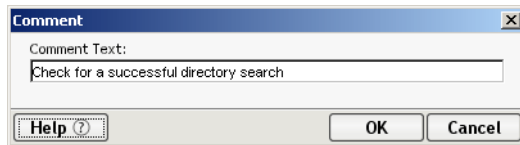
9 Highlight the Execute DSML Request statement in the Action Model.

EXERCISE 7-2: Add a comment to the Action Model.

1 RMB>New Action>Comment.




- 2 Enter **Check for a successful directory search** in the Comment Text field.



- 3 Click **OK**.

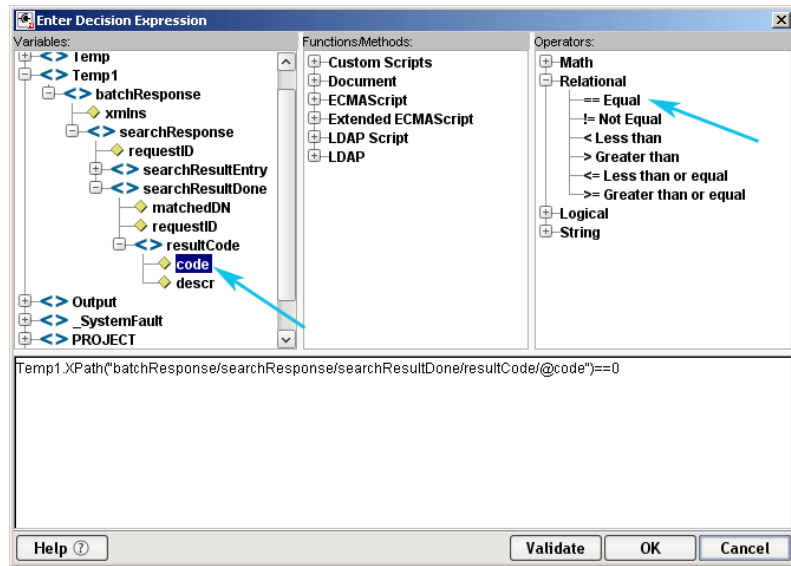
EXERCISE 7-3: Add a Decision Action

The Decision Action you add will check for a successful directory search and branch on that result. If true, the component will execute map actions that will map data to the Output Part. If false, the component will log a message to System Out.

- 1 Highlight the comment “**Check for a successful directory search**” in the Action Model.
- 2 **RMB>New Action >Decision**.
The Decision Dialog appears.
- 3 Click the **Editor Expression** Button on the Dialog. 
The Enter Decision Expression Dialog appears.
- 4 In the Variables pane of the dialog drill down the tree to the following node: **Temp1/batchResponse/searchResponse/searchResultDone/resultCode**. Click on the **plus** symbol next to each of the respective nodes to expand them.
- 5 Double-click on the **code** element under the **resultCode** node.

The XPath expression is added to the editor pane.

- 6 Expand the **Relational** node in the Operators pane.
- 7 Double-click on the == **Equal** element.
- 8 Click in the lower pane of the dialog just after the equal signs, type **0**.



The expression,

Temp1.XPath("batchResponse/searchResponse/searchResultDone/resultCode/@code")==0 appears in the lower window of the dialog.

- 9 Click **OK** to close the expression builder.



- 10 Click **OK** to close the Decision Dialog.

Note the Decision Action is added to the Action Model.

If you examine the syntax, you will see the path through the tree structure to the code element.

- ◆ Temp.XPath("batchResponse/searchResponse/searchResultDone/resultCode/@code")
- ◆ Temp is the Part
- ◆ batchResponse is the root node
- ◆ searchResponse is a child node of the root
- ◆ searchResult Done is a child node of searchResponse
- ◆ resultCode is a child node of searchResultDone
- ◆ @code indicates retrieve the data from the element code

EXERCISE 7-4: Add Map Actions

- 1 Highlight the **TRUE** branch of the Decision Action.
- 2 Expand the **nodes** in the Temp1 part under searchResponse to expose the attributes retrieved by the search.

The attribute nodes are abbreviated **attr**. You may have to use the scroll bar on the side of the dialog to access all of the nodes.


Temp1	Data
batchResponse	urn:oasis:names:tc:DSML:2
xmlns	
searchResponse	
requestID	65
searchResultEntry	
dn	cn=SSpade,ou=HR,o=newell
requestID	65
attr	
name	title
value	Chief of Pediatric Surgery
name	sn
value	Spade
name	mail
value	sspade@megahospital.org
name	telephoneNumber
value	555-667-7598

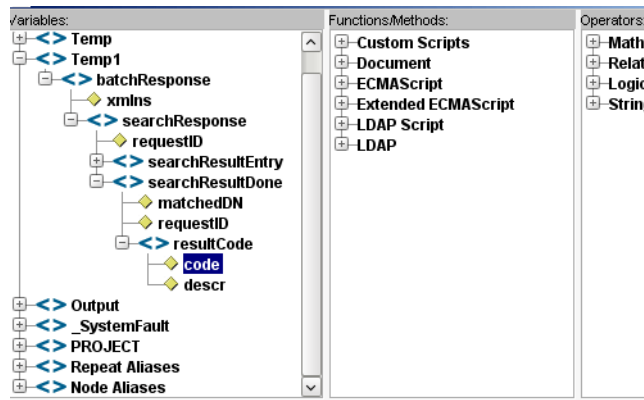
- 3 Use drag and drop to map the Temp1 Part **attr/value** elements **sn**, **title**, **mail** and **telephoneNumber** to **physician**, **Department**, **email**, and **phone**, respectively in the Output Part. Be careful to drag the **value** element under **attr**, in order to properly retrieve the data.

Note the map actions have been added to the “**TRUE**” branch of the Decision Action in the Action Model.

EXERCISE 7-5: Add a Log Action

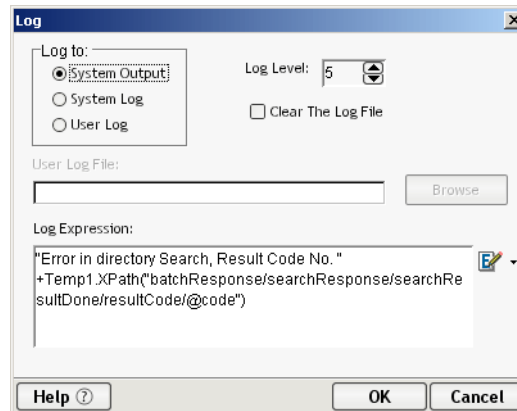
You will be adding two log actions that will output messages to System Out. The log actions will output the result code and description from the directory search, if the search was not successful.


- 1 Highlight the **FALSE** branch of the Decision Action.
- 2 **RMB>New Action>Log.**
- 3 Enter ("**Error in directory Search, Result Code No.** +") in the Log Expression text window, include the quotes, the space after No., and the plus sign.
- 4 Click on the **Expression Editor** button next to the text window. 
- 5 Expand the **Temp1** node in the Variables pane to access the **code** attribute under the resultCode element.
- 6 Double-click on the **code** attribute to place the expression in the editor text pane.

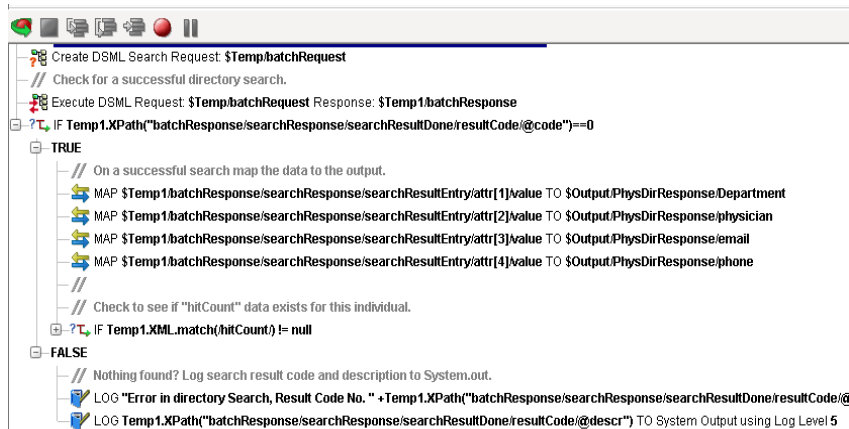


"Error in directory Search, Result Code No. "
+Temp1.XPath("batchResponse/searchResponse/searchResultDone/resultCode/@code")

- 7 Click **OK.**

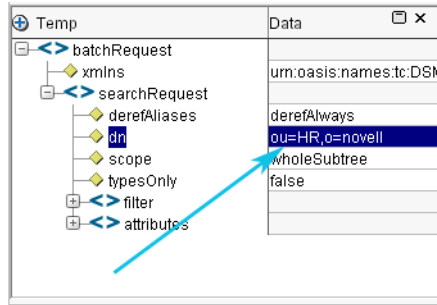


- 8 Click **OK** to add the log expression to the Action Model.
 - 9 Highlight the **Log** action you just added in the Action Model.
 - 10 **RMB>New Action>Log**.
 - 11 Click on the **Expression Editor** button next to the text window. 
 - 12 Expand the **Temp1** and its child nodes in the Variables pane to drill down to the **descr** attribute under the **resultCode** node.
 - 13 Double-click on the **descr** attribute to place the expression in the editor text pane.
 - 14 Click **OK**.
 - 15 Click **OK** to add the log expression to the Action Model.
- Your Action Model should look similar to the one pictured.

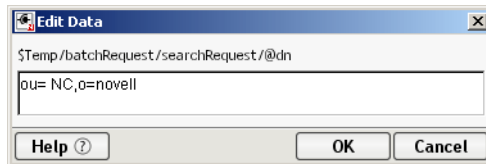


Execute your component. You will see data mapped to the Output part.

- 16 To execute the False path of the Decision Action, select **Animate>Start Animation** from the Integration Manager toolbar.
 - 17 Using the **Step Over** button on the animation toolbar, click **twice** to animate to the **Execute DSML** statement in the Action Model.
- The Temp Part is now populated with the search request data.
- 18 In the **Temp Part** expand the **searchRequest** node, and double-click on the **Data** in the **dn** element.



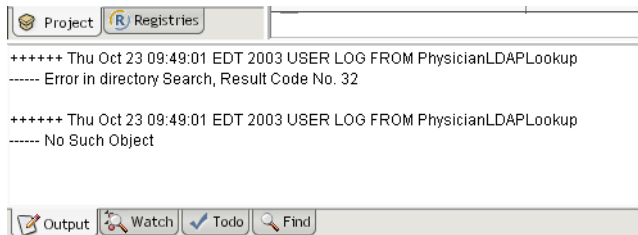
19 In the Edit Data dialog, modify the expression `ou=HR` to `ou=NC`.



20 Click **OK**. The data in the Temp Part is changed. (This will cause the LDAP search to fail since NC is not in the directory.)

21 Click the **Run to Breakpoint End** button on the Animation Toolbar.

The Log messages will appear in the Message Pane Output tab.



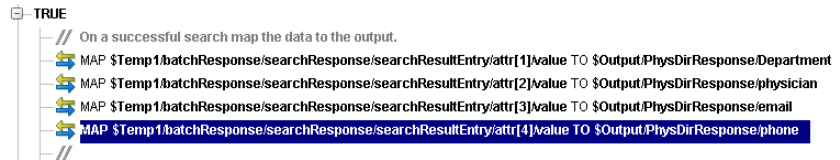
22 **File>Save** to save your work.

EXERCISE 7-6: Create or increment the hit count for the physician


The *hitCount* element has a value of null when the directory is first created. In the next set of exercises you will use a decision statement to determine if the hitCount element has a value in the directory. If the hitCount element exists, you will return the current hit count in the output, increment the count, and update the directory. If the element does not exist you will modify the directory to include the element, giving it an initial value of one.

The decision statement will utilize an Integration Manager extension method “XML”. The XML method returns a string representation of the DOM it receives as input, in this example Temp1. You will then use the ECMAScript string match method to search for the string “hitCount”, within the string returned from the XML method. If a match is found, the True branch of the decision statement will be executed. If no match is found, the False branch will be executed.

- 1 Highlight the last **Map** action in the True branch in the action model.



- 2 RMB>New Action>Decision.

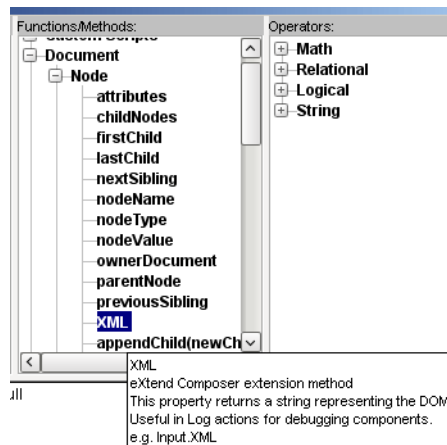
- 3 Click on the **Expression Editor** button next to the text window. 

- 4 Double-click on **Temp1** in the Variables pane in the editor.

- 5 Expand the **Document** element in the Functions/Methods pane in the editor.

- 6 Expand the **Node** element under the Document element.

- 7 Double-click on **XML** under the Node element.



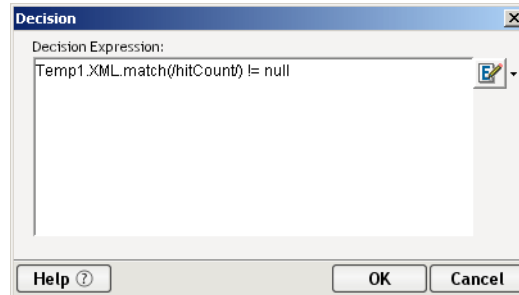
- 8 Scroll down in the Functions/Methods window and expand the **ECMAScript** element.

- 9 Locate and expand the **String** element under the ECMAScript element.

- 10 Double-click on the **match** method under the String element.

The lower pane of the editor should now contain the text:
 “Temp1.XML.match()”.

- 11** Click between the parenthesis following the word match, and enter “/hitCount”, do not include the quotes.
- 12** Click after the ending parenthesis and enter “!=null”.
The lower pane of the editor should now contain the text:
“Temp1.XML.match(/hitCount/) != null”.
- 13** Click **OK**.
Your decision dialog should look similar to the one depicted below.



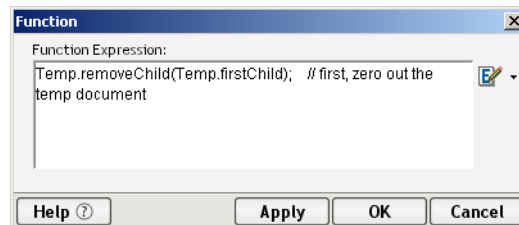
- 14** Click **OK**.
- 15** Save your work, select **File>Save** from the menu.

EXERCISE 7-7: Add Actions to the False Branch of the Decision action.

In this exercise you will add the actions to create the hit count, if it is not currently part of the physician directory entry. First you need to add actions that will clear the temp documents, to create a clean scratch pad area.

- 1** Highlight the **False** branch of the decision statement you just added.
- 2** **RMB>New Action>Function**.
- 3** Enter the text (without quotes) “**Temp.removeChild(Temp.firstChild); // first, zero out the temp document**” in the Function Expression text window.
- 4** Click **OK**.

You could, alternatively, use the expression editor to create the statement. Make sure you include the semi-colon after the last parenthesis.



You will use the `removeChild` function a second time to clear the `Temp1` document.

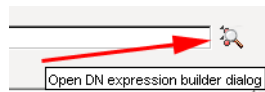
- 5 Highlight the **Temp.removeChild** function action you just added.
- 6 **RMB>New Action>Function.**
- 7 Enter the text “**Temp1.removeChild(Temp1.firstChild);**” in the Function Expression text window.
- 8 Click **OK.**

Next you will create the DSML statements to add the `hitCount` element to the `physician`, whose `cn` (common name) element in the directory matches that of the one passed into the Input part of the LDAP component.

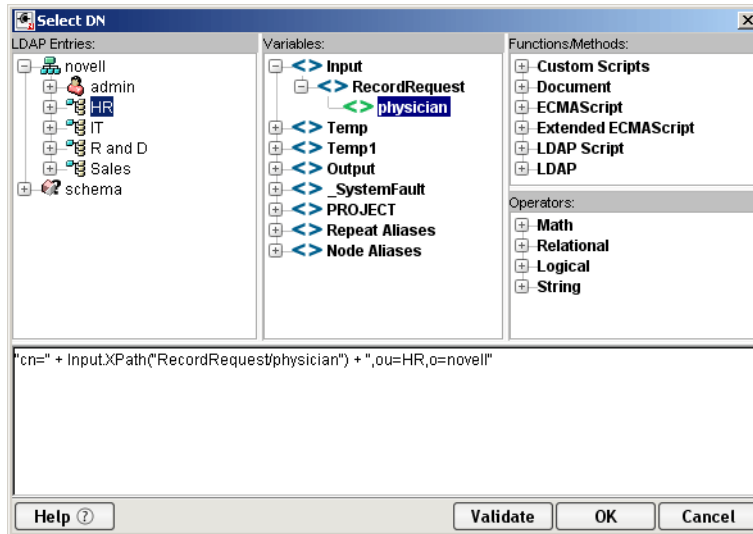
- 9 Highlight the **Temp1.removeChild** function call in the action model.
- 10 **RMB>New Action>Create DSML.**
- 11 Select **Modify** from the DSML Action dropdown list.
- 12 Select **Temp** from the Request Map dropdown list.
Accept the default `batchRequest` in the Request Map text field.
- 13 Click **OK.**

The modify action will add the `hitCount` element to the directory entry for the `physician`. Steps 14 through 21 set up the XPath expression that maps the proper `physician` element in the directory.

- 14 In the native environment pane, enter “**cn=**” + (include the quotes and the plus sign) in the Entry DN text field of the Modify tab.
- 15 Click the **DN expression builder** button.



- 16 In the Variables pane of the expression builder, expand the **Input** node.
- 17 Expand the **RecordRequest** node in the Input tree.
- 18 Double-click on the **physician** element.
The text “`Input.XPath("RecordRequest/physician")`” is added to the expression in the lower pane of the expression builder.
- 19 Enter a **plus sign** “+” after the second parenthesis in `physician`.
The expression should read: `"cn=" + Input.XPath("RecordRequest/physician") +`
- 20 Expand the **novell** node in the LDAP entries window.
- 21 Double-click on the **HR** node in the novell tree.
The expression should read: `"cn=" + Input.XPath("RecordRequest/physician") + ",ou=HR,o=novell"`.



22 Click **OK**.

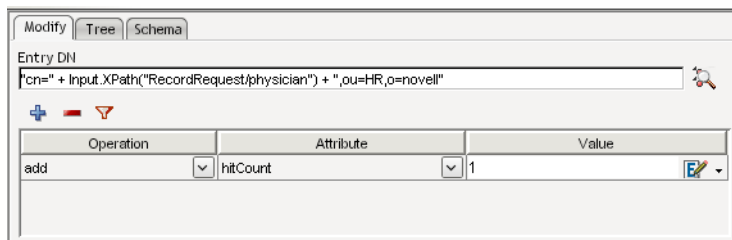
Use the add operation to add the hitCount element with an initial value of 1.

23 In the native environment Modify tab, click the **plus** button to add a new row to the attribute modification mappings.

24 Select **add** from the Operation dropdown list.

25 Select **hitCount** from the Attribute dropdown list.

26 Enter the number **1** in the Value field.



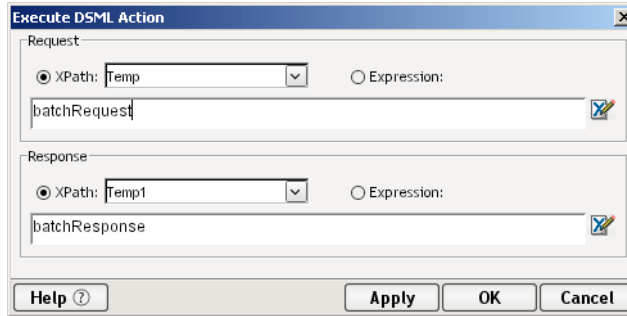
27 Highlight the **Create DSML Modify Request** in the LDAP component action model.

28 **RMB>New Action>Execute DSML**.

29 Select **Temp** from the Request part dropdown list.

30 Select **Temp1** from the Response part dropdown list.

Accept the default values for the remaining dialog fields.



31 Click **OK**.

The Execute DSML Request is added to the action model. Map the newly created data value for hitCount to the NoOfInquiries attribute in the Output part.

32 In the action model, highlight the **Execute DSML Request** that you added in the preceding steps.

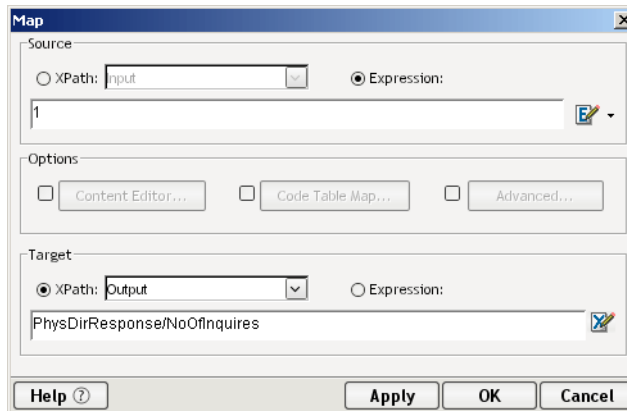
33 **RMB>New Action>Map**.

34 Select the **Expression** radio button in the Source section of the dialog.

35 Enter the number **1** in the Source section text field.

36 Select the **XPath** radio button in the Target section of the dialog.

37 Enter **PhysDirResponse/NoOfInquires** in the Target text field.



38 Click **OK**.

39 Select **File>Save** from the menu to save your work.

40 Select **Component>Execute** from the menu.

Your False block for the if hitCount decision action should look similar to the one below.

NOTE: Comments have been added to the tutorial action model to clarify the model.

```

FALSE
// No hitCount data? Initialize the person's hitCount to 1 in the directory;
f(x) CALL Temp1.removeChild(Temp1.firstChild); //first, zero out the temp document
f(x) CALL Temp1.removeChild(Temp1.firstChild); // zero out this one, too
Create DSML Modify Request: $Temp1.batchRequest
Execute DSML Request: $Temp1.batchRequest Response: $Temp1.batchResponse
// Map the data value to Output
MAP 1 TO $Output/PhysDirResponse/NoOfInquires

```

EXERCISE 7-8: Add Actions to the True Branch of the Decision action.

In this exercise you will add the actions to increment the hit count, if it is currently part of the physician directory entry.

Before incrementing the count, map the current value of the count to the output. To accomplish this you need to add a breakpoint to the action model. Breakpoints are handy for debugging and modifying components during development. In this case, you will use the breakpoint to assist with mapping the hitCount to the Output part.

- 1 Highlight the first **Decision** action in the model. The one that checks for a successful search.
- 2 Click the **Toggle Breakpoint** button on the animation toolbar.





The Decision action will become red indicating that there is a breakpoint set on that action.

```

Execute DSML Request: $Temp1.batchRequest Response: $Temp1.batchResponse
-?T IF Temp1.XPath("batchResponse/searchResponse/searchResultDone/resultCode/@code")=-0
TRUE

```

- 3 Click the **Start Animation** button on the animation toolbar. 
- 4 Click the **Run to Breakpoint/End** button on the animation toolbar. 

The results from the physician search are now in the Temp1 part. To facilitate the mapping of the hitCount, reload the XML document for the Output part. If you have forgotten how to do this, refer back to EXERCISE 7-1: “Set up the component for edits”.

- 5 Highlight the True branch within the check for match decision action.

```

-?T IF Temp1.XML.match(/hitCount) != null
TRUE

```


- 6 Expand the nodes in the Temp1 part under **searchResponse** to expose the attributes retrieved by the search.
- 7 Drag the **value** element of the **hitCount** attribute and drop it on the **NoOfInquiries** element in the Output part.

Once the hitCount has been mapped, the component must increment and update the value in the directory. The component will use function calls to clear the Temp parts, to create a clean scratch pad area that will hold the modify request and response documents. You can do this by using cut and paste from the False block you created in the previous exercise.

- 8 Highlight the **Temp.removeChild(Temp.firstChild)** function call in the False block created in the previous exercise.
- 9 Holding down the shift key, click on the second function call in the block, **Temp1.removeChild(Temp1.firstChild)**.
Both function calls should now be highlighted.

```

- FALSE
  // No hitCount data? Initialize the person's hitCount to 1 in the directory:
  f(x) CALL Temp.removeChild(Temp.firstChild); // first, zero out the temp docum
  f(x) CALL Temp1.removeChild(Temp1.firstChild); // zero out this one, too

```

- 10 RMB>Copy.

- 11 Highlight the **Map hitCount** action in the True block under the Temp1.XML.match decision action.

- 12 RMB>Paste.

The function calls are added to the action model.

```

- ? IF Temp1.XML.match(/hitCount/) != null
  TRUE
    // If hitCount exists, map the data to Output
    MAP $Temp1/batchResponse/searchResponse/searchResultEntry/attr[5]/value
    // ... and bump the hit count in the directory
    f(x) CALL Temp.removeChild(Temp.firstChild); // first, zero out the temp docum
    f(x) CALL Temp1.removeChild(Temp1.firstChild); // zero out this one, too

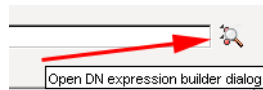
```

Add the DSML Create and Execute statements to the True block.

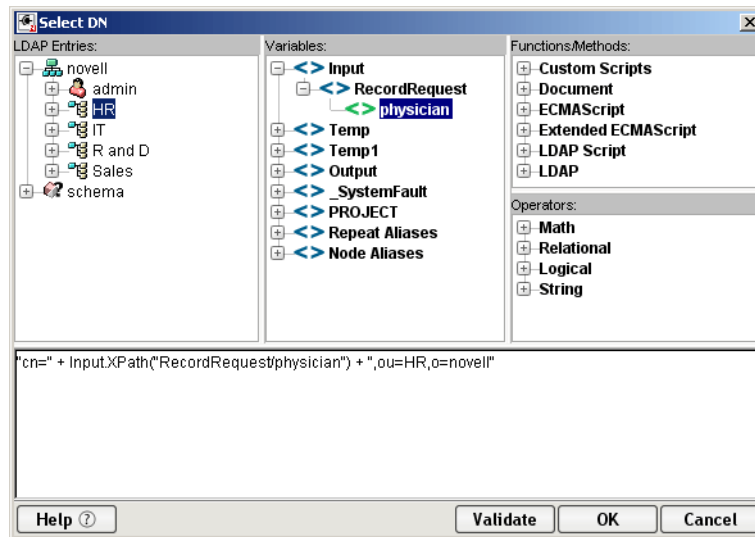
- 13 Highlight the **Temp1.removeChild** function call in the True block.
- 14 RMB>New Action>Create DSML.
- 15 Select **Modify** from the DSML Action dropdown list.
- 16 Select **Temp** from the Request Map dropdown list.
Accept the default batchRequest in the Request Map text field.
- 17 Click **OK**.

Set up the XPath expression that maps to the proper physician element in the directory.

- 18 In the native environment pane, enter “cn=” + (include the quotes and the plus sign) in the Entry DN text field of the Modify tab.
- 19 Click the **DN expression builder** button.



- 20 In the Variables pane of the expression builder, expand the **Input** node.
- 21 Expand the **RecordRequest** node in the Input tree.
- 22 Double-click on the **physician** element.
The text “Input.XPath("RecordRequest/physician")” is added to the expression in the lower pane of the expression builder.
- 23 Enter a **plus sign** “ + “ after the second parenthesis in physician.
The expression should read: "cn=" + Input.XPath("RecordRequest/physician") +
- 24 Expand the **novell** node in the LDAP entries window.
- 25 Double-click on the **HR** node in the novell tree.
The expression should read: "cn=" + Input.XPath("RecordRequest/physician") + ",ou=HR,o=novell".



- 26 Click **OK**.
Use the replace operation to increment the hitCount element.
- 27 In the native environment Modify tab, click the **plus** button to add a new row to the attribute modification mappings.
- 28 Select **replace** from the Operation dropdown list.
- 29 Select **hitCount** from the Attribute dropdown list.

- 30** Click the **Expression builder** button next to the Value field.

The current hitCount is available in the Output part as NumberOfInquiries. You can use this value to modify the directory entry. XML documents are represented as strings. In order to increment the hitCount, you must cast the value to a Number. You will be using an ECMAScript function to perform the cast.

- 31** Enter **Number()** in the lower pane of the Expression builder.

- 32** Click between the parenthesis following the word **Number**.

Use the pick list in the variables window to select the NumberOfInquiries element from the Output.

- 33** In the Variables pane expand the **Output** node.

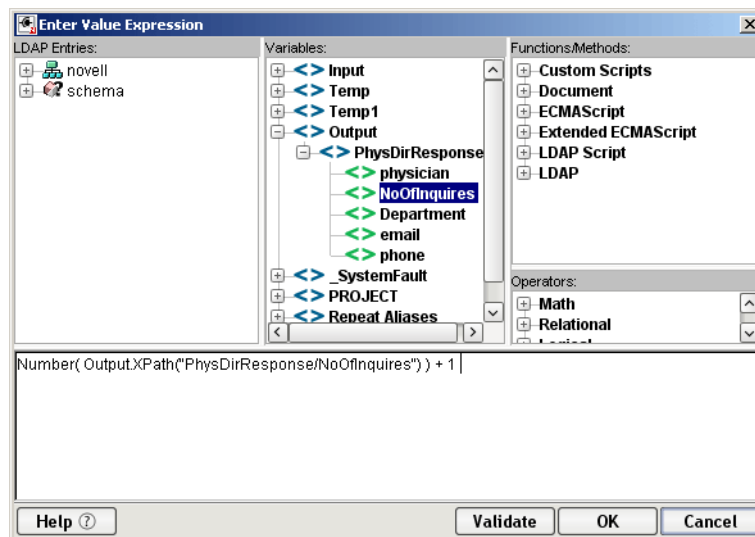
- 34** Expand the **PhysDirResponse** node under Output.

- 35** Double-click on the **NumberOfInquiries** element under PhysDirResponse.

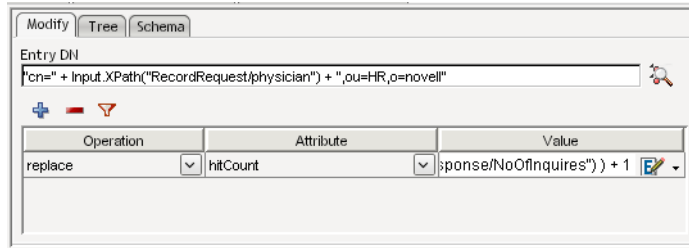
The XPath expression `Output.XPath("PhysDirResponse/NoOfInquires")` is placed between the parenthesis.

- 36** Enter `+ 1` (no quotes) after the last parenthesis in the editor pane.

The expression `Number(Output.XPath("PhysDirResponse/hitCount")) + 1` should now appear in the lower window of the editor. The expression editor should look similar to the one pictured.



- 37** Click **OK**.



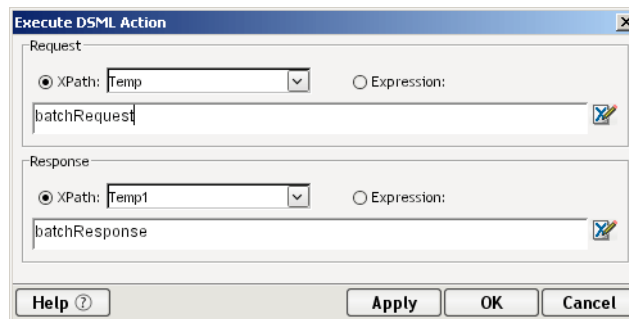
38 Highlight the **Create DSML Modify Request** that you added in the action model.

39 **RMB>New Action>Execute DSML.**

40 Select **Temp** from the Request part dropdown list.

41 Select **Temp1** from the Response part dropdown list.

Accept the default values for the remaining dialog fields.



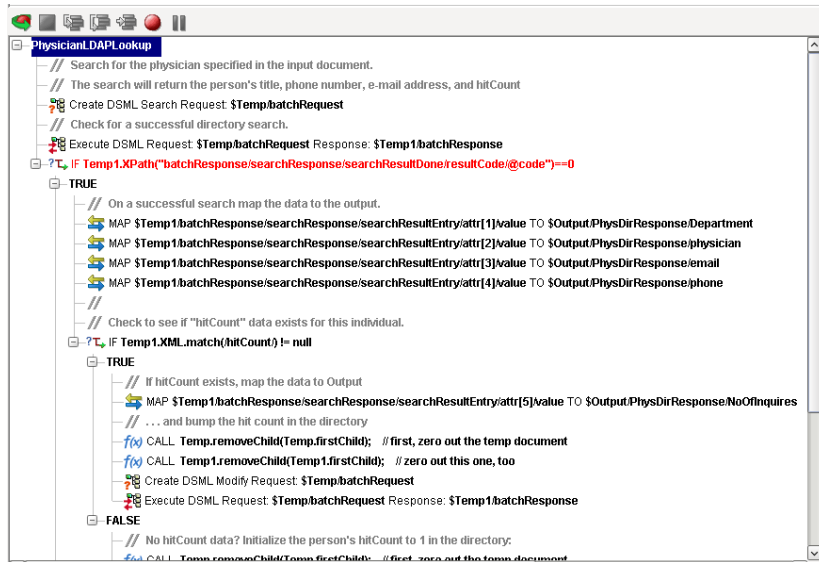
42 Click **OK**.

The Execute DSML Request is added to the action model.

43 Select **File>Save** from the menu.

44 Select **Component>Execute** from the menu.

By executing the component several times you can observe the NumberOfInquires increment in the Output part. Your component action model should be similar to the one pictured.



Edit the PatientLookup XML Map Component

The web service described in Lesson 1 required that the output from the patient database and physician directory, be combined prior to returning the data to the consumer. (Requirement number 5)

In this series of exercises, you will edit the PatientLookup component. The component will call both the JDBC and LDAP components you have created, and combine their output. The final output of the PatientLookup component will be passed to the web service

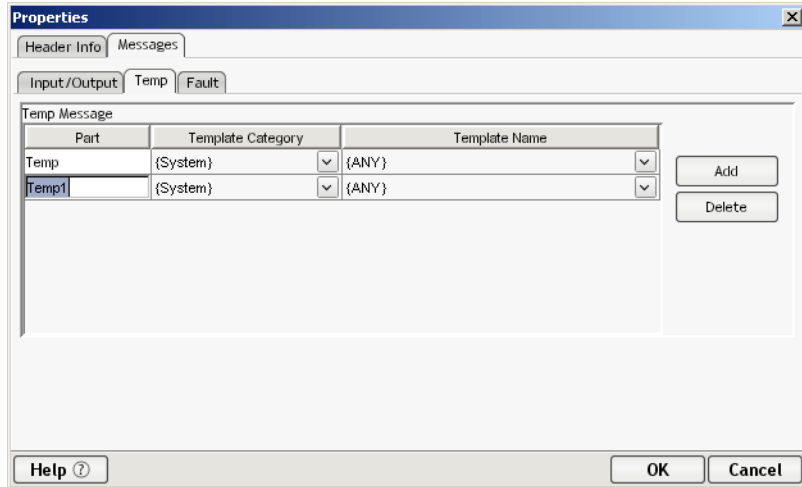
EXERCISE 7-9: Edit the XML map component properties

In this exercise you will modify the properties of the component adding two temp parts to use as scratch pads, while combining the data for output. In the Category pane under Components, highlight **XML Map**.

- 1 Highlight **PatientLookup** in the Instance pane.
- 2 **RMB>Properties**
- 3 Click on the **Messages** tab in the Properties dialog.
- 4 Select the **Temp** tab.
- 5 Press the **Add** button **twice**.

Two parts, Temp and Temp1, are added to the dialog.

Leave the Template Category and Template Name at the default values.



- 6 Click OK.

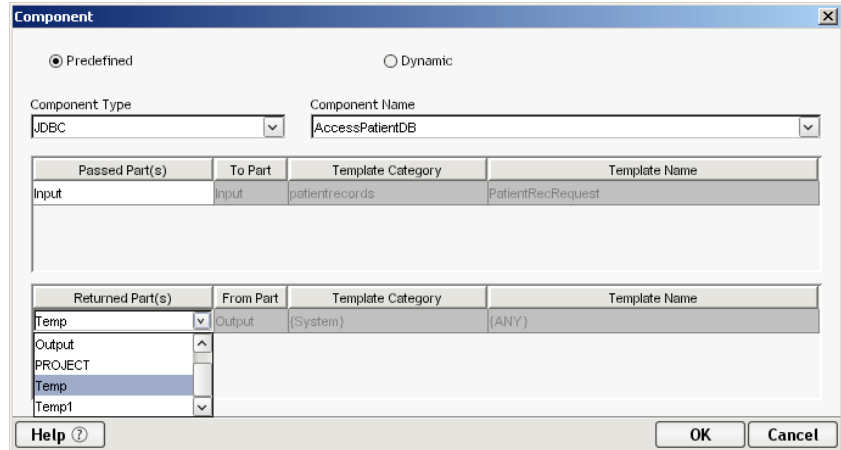
EXERCISE 7-10: Add Component Actions

Currently the PatientLookup component maps the name of the physician from the input to the output. You will first need to delete this map action. You will then add other actions to the model.

- 1 Select **XML Map** in the Category pane.
- 2 Double-click on **PatientLookup** in the Instance pane to open the component.
- 3 Click on the **Map** action in the Action Model.
- 4 **RMB>Delete.**

A confirmation dialog will appear, with the text “Are you sure you want to delete this action?”

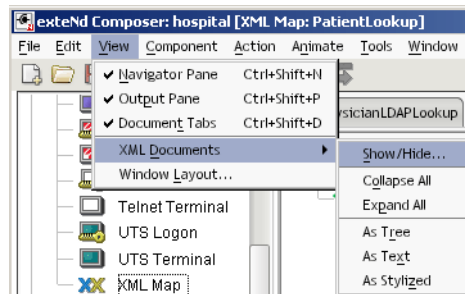
- 5 Click the **Yes** button on the Confirm Delete dialog.
- Add the actions that will call the JDBC and LDAP components.
- 6 Highlight **PatientLookup** in the Action Model.
- 7 **RMB>New Action>Component.**
- 8 Select **JDBC** from the Component Type dropdown list.
- 9 Select **AccessPatientDB** from the Component Name dropdown list.
- 10 Select **Temp** from the dropdown list under Returned Part(s).



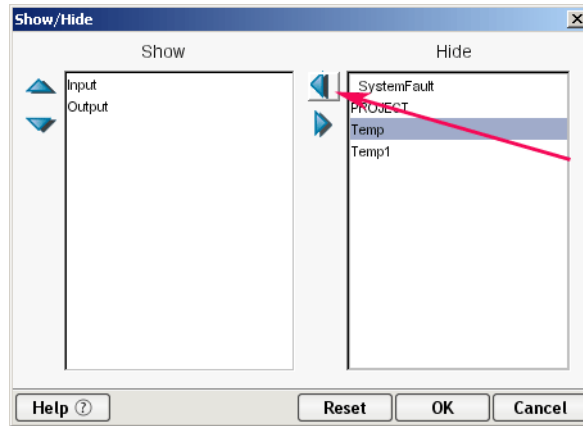
- 11 Click **OK**.
- 12 Highlight the **Execute AccessPatientDB** action in the Action Model.
- 13 **RMB>New Action>Component**.
- 14 Select **LDAP** from the Component Type dropdown list.
- 15 Select **PhysicianLDAPLookup** from the Component Name dropdown list.
- 16 Select **Temp1** from the dropdown list under Returned Part(s).
- 17 Click **OK**.

Two temp parts should now appear in the Native Environment pane with the Input and Output parts. If they do not appear, use the Show/Hide dialog from the menu to display them.

- 18 Select **View>XML Documents>Show/Hide** from the menu.



- 19 Highlight the **Temp** part in the Hide window.
- 20 Click the **left pointing arrow** button between the Show and Hide windows to move the document to the show list.

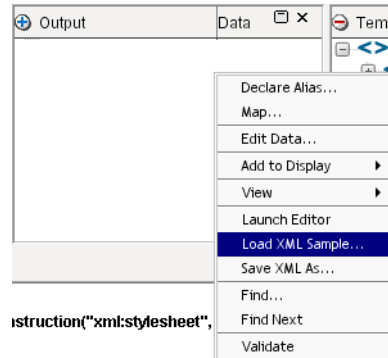


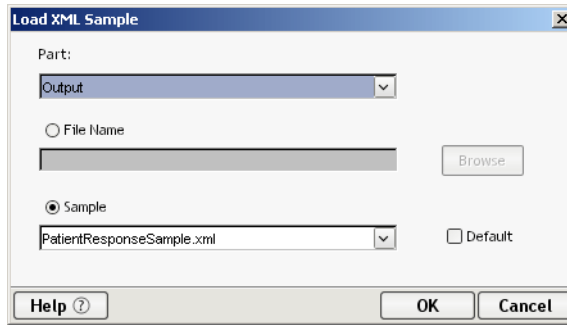
21 Repeat steps 18 and 19 for the **Temp1** part.

EXERCISE 7-11: Add Map Actions

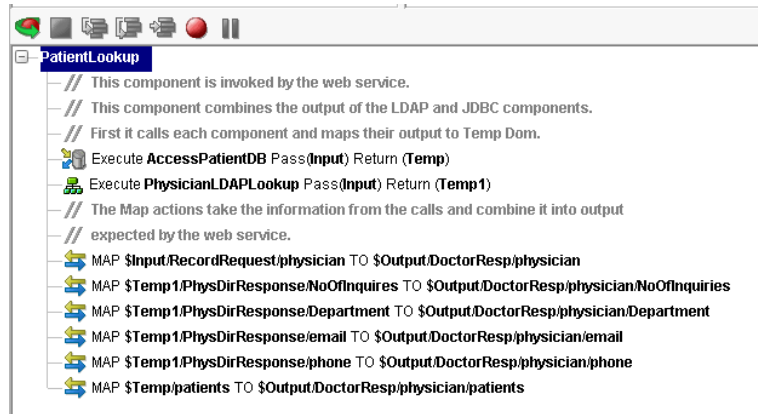
To complete the mapping, you will first have to execute the component. This will provide you with data, from the directory and the database to use in mapping. After executing the component, you will need to reload the sample document in the Output part.

- 1** Select **Component>Execute** from the menu.
- 2** Click **OK** on the Component Executed message dialog.
- 3** Click in the **Output Part**.
- 4** **RMB>Load XML Sample.**





- 5** Click **OK** on the Load XML Sample dialog.
- 6** Expand the **physician** node in the Output part.
Using drag and drop, you will map the results from the two component actions to the output. Note that each drag and drop creates a new map action in the Action Model.
- 7** Highlight the **Execute PhysicianLDAPLookup** action.
- 8** Drag the **NoOfInquiries** element from the Temp1 part to the **NoOfInquiries** element in the Output.
- 9** Drag the **Department** element from the Temp1 part to the **Department** element in the Output.
- 10** Drag the **email** element from the Temp1 part to the **email** element in the Output.
- 11** Drag the **phone** element from the Temp1 part to the **phone** element in the Output.
- 12** Drag the **patients** node (this will map all the child nodes of this node) from the Temp part to the **patients** node in the Output part.
- 13** Highlight the **Execute PhysicianLDAPLookup** action.
- 14** Drag the **physician** element from the Input part to the **physician** element of the Output part.
Your action model should be similar to the one pictured here.



15 Select **File>Save** from the menu.

16 Select **Component>Execute** from the menu.

Summary of what you've done

You have accomplished the following tasks:

- ◆ Learned about the Integration Manager Action Model and Actions:
- ◆ Have been briefly introduced to XPath, ECMAScript and the Integration Manager Expression Builder
- ◆ Worked with Actions
 - ◆ Created a Comment Action
 - ◆ Created a Decision Action
 - ◆ Created a Log Action
 - ◆ Created a Function Action
 - ◆ Created a Component Action

Next lesson

In the next lesson you will learn about publishing and consuming web services.

8

Publishing and Consuming Web Services

What you will learn

In this lesson you will complete the last outstanding requirement outlined for your Integration Manager project in the first lesson. You will learn how to trigger your service in a number of ways. You will also act as a consumer of your service via XML and Web Services Interchange. First review the requirement.

The service may be invoked by any of the following service triggers:

- ◆ Servlet of the type Params(URL/Form) that outputs a raw XML document
- ◆ Servlet of the type Params(URL/Form) that uses an XForm
- ◆ Servlet that utilizes a JSP to format the output
- ◆ Servlet of the type XML (HTTP/Post)
- ◆ SOAP Servlet using WSDL

After creating and deploying the service with the service triggers, you will act as a consumer of the service via both XML Interchange, and Web Service Interchange.

What you will do

You will perform the following exercises

- ◆ Create a JSP resource file
- ◆ Add the JSP to the Deployment Component
- ◆ Deploy the Web Service with the JSP
- ◆ Create an XML Map Component that uses XML interchange
- ◆ Modify the Deployment Component for the XML Interchange.

- ◆ Deploy the Web Service with Servlet Type XML(HTTP/Post)
- ◆ Create a WSDL for the PatientRecResponse Web Service
- ◆ Create XML Templates for the WSDL
- ◆ Add a Soap Trigger to the Deployment Component
- ◆ Deploy the Web Service with SOAP
- ◆ Create an XML Map component that executes a WSInterchange action
- ◆ Create an XForm for Physician Input.
- ◆ Deploy the XForm as a web service

How long will it take? About 40 minutes

Publishing Web Services

Servlets

The most common analogy regarding servlets is that they are to servers, what applets are to browsers. Servlets model a request/response architecture. A client makes a request of the server and the server provides a response. The servlet acts a middle-tier between the client and the server, containing the information necessary to invoke the service. A servlet resides on the server where the web service is deployed. When a hit on the web services URL reaches the server, the servlet triggers the service. In lesson 5 you deployed the service using a servlet of type Params(URL/Form). The servlet type is based on the method by which it receives parameters. In this lesson you will be creating a JSP resource to use with the Params(URL/Form) servlet. You will also create additional servlets of type SOAP and XML (HTTP/Post).

Web Service Description Language (WSDL)

After designing, developing, and testing a web service, the next step is to publish it via the UDDI (Universal Description, Discovery and Integration) registry. Publishing the service allows potential business partners to discover and utilize the service. A WSDL file is an XML file that contains all the necessary information to invoke your service. There are many sources of information on WSDL, including the SUN website. <http://java.sun.com/>

Integration Manager generates the WSDL files for your service. In this lesson you will generate a WSDL file for the PatientReqRequest web service. The SOAP servlet you create will be based on this file.

Consuming Web Services

Java Server Pages (JSP)

A complete explanation of JSP is outside the scope of this tutorial. The Sun website offers a vast amount of information, and a tutorial on JSP.

<http://developer.java.sun.com/developer/onlineTraining/JSPIntro/>

JSP provide a means of separating the business logic of a service, from the presentation logic of the browser. By separating display issues from business logic, JSP provide platform independence for web services. Several different web pages or browsers, can invoke a service via a JSP. The web page can then format the data returned by the service in a manner suitable for their audience.

For the purpose of the tutorial, you will be creating a JSP resource within Integration Manager. The page will invoke your web service and publish the output to the browser. The tutorial's JSP scenario represents a business to client exchange between the JSP and the web service.

XForms

Forms are the most common way of retrieving data from a web browser. An example of a form might be the order page for a book store site. XForms separate form data from the presentation logic of the device on which the “form” is displayed. They differ from JSP in that XForms are not limited to use by a server.

XForms provide device independence, the same form might collect data from a hand held device, a phone, or a browser window. For example, a dropdown list in a browser window might appear as a selection on a menu in a handheld device. The XForm would process the data from either into the same XML document and pass the data on to the web service. This separates the need for the service to have knowledge of the input device.

XForms process the “instance data” into an XML document. The data is then passed on to the server. The XForm maps the data to and from the form controls of the display device. The instance data is defined using an XPath tree representation. This method of structuring the data “binds” the data to the form control. The XForm standard defines the form controls For more information on XForms refer to the W3C's website.

<http://www.w3.org/MarkUp/Forms>.

In the tutorial you will create an XForm that will utilize a textfield for entry. You will create a web service that will invoke the XForm processor and display the form in the browser. When the user clicks the submit box on the form, the data in the form will be retrieved and the PatientRecReq web service will be invoked.

XML Interchange

The XML Interchange action in Integration Manager reads external XML documents into a component and writes the data out as XML files. There are four types of XML Interchange actions: get, put, post and post with response.

You will be using the post with response action in the tutorial. The XML Interchange you create represents a business to business exchange. A business that utilized your web service might “post” its input to the web service as an XML document, via the interchange. The web service would then process the document and return a response.

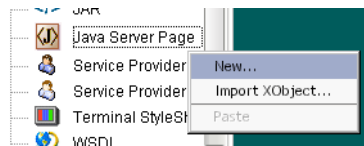
WS Interchange

The WS Interchange is another example of a business to business exchange. As discussed in the above WSDL section, the web service would publish a WSDL describing the service. A business searching for services via the UDDI registry, would obtain a copy of the WSDL, and generate the appropriate SOAP trigger for the service. The business would then utilize the WSInterchange action to invoke the service with the SOAP servlet.

NOTE: The LDAP directory evaluation tool, the MySQL service and the application server must all be running on your system to successfully complete the lab exercises for this unit.

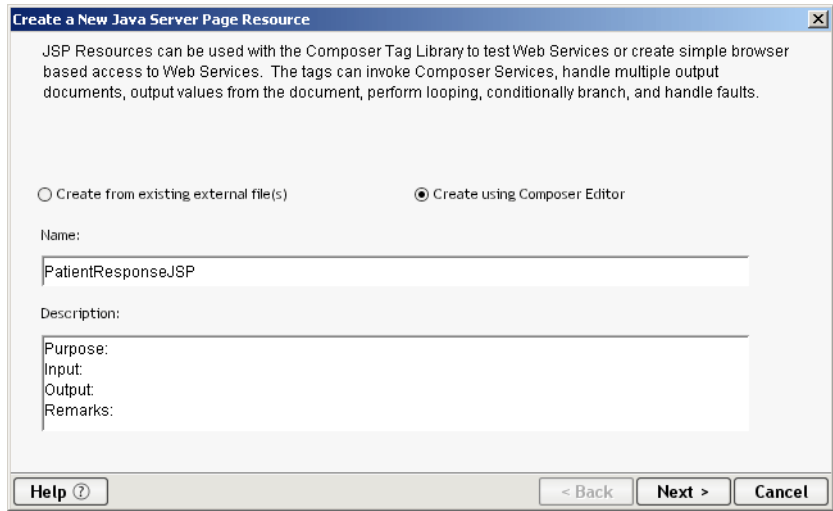
EXERCISE 8-1: Create a JSP resource file

- 1 If Integration Manager is not running on your system, launch Integration Manager as described in Exercise 1-1.
- 2 If you are not already in the “**hospital**” project, open it by either selecting it from the recent project list **File>Recent** on the menu, or browse to the project location by selecting **File>Open Project**
- 3 Select **Java Server Page** under Resources in the Category Pane.
- 4 Select **JSP** from the Resource section of the Navigation frame.
- 5 **RMB>New.**



The Create a New Java Server Page Resource wizard appears.

- 6 Select the **Create using Composer editor** radio button.
- 7 Enter **PatientResponseJSP** in the name text field.



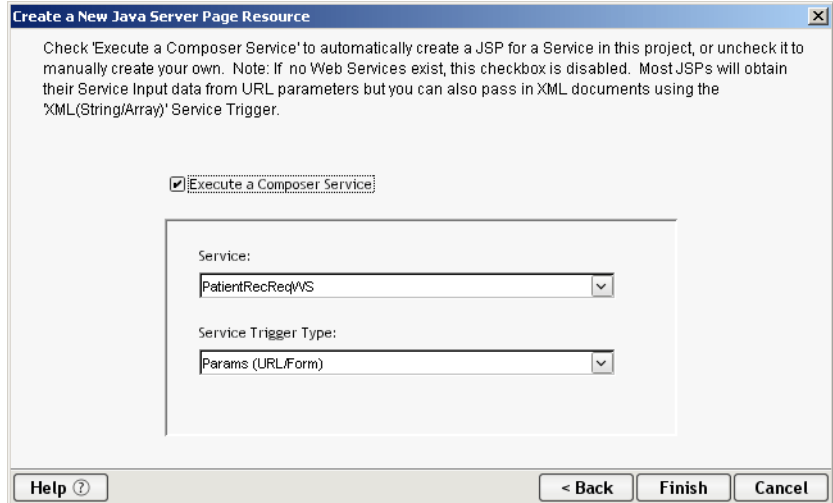
8 Click **Next**.

9 Select the **Execute a Composer Service** checkbox.

This will cause the Integration Manager wizard to generate a JSP file with the proper tags to invoke your service.

10 Select **PatientRecReqWS** from the Service drop down list.

11 Select **Params/URL** as the Service Trigger Type.



12 Click **Finish**.

A **PatientResponseJSP** is created and opened in the Integration Manager Native Environment window. The **PatientResponseJSP** is added to the Instance pane.

You will have to edit the JSP to tell it the deployment object of your service.

- 13 Find the `<composer:execute service=` tag in the file.
- 14 Enter **GetRecords.** after the `.` following the word `hospital` and preceding `PatientRecReqWS`.

```

<%@ taglib uri="/composer" prefix="composer" %>

<composer:execute service="composer.hospital.GetRecords.PatientRecReqWS" %>

<html>
<head>
  <title>
    PatientRecReqWS
  </title>
</head>

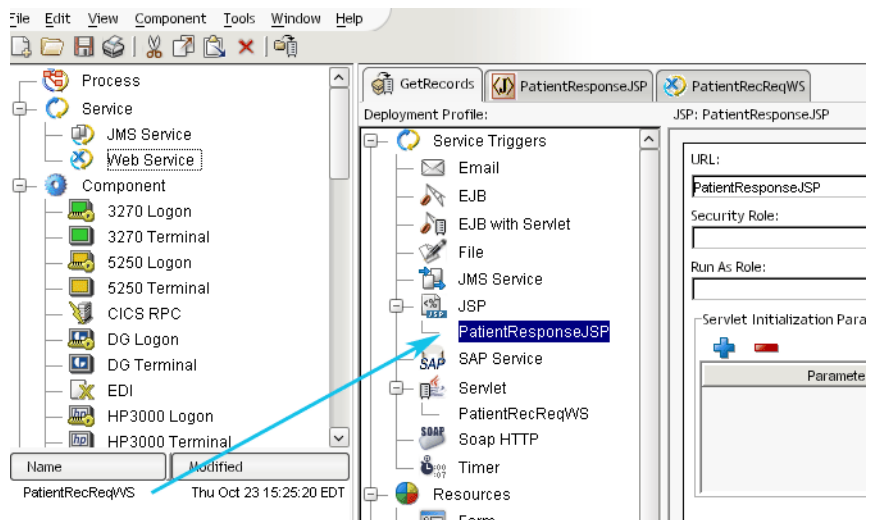
```

The line should now look like this one indicated by the arrow in the image above.

- 15 Select **File>Save** from the menu to save your work.

EXERCISE 8-2: Add the JSP to the Deployment Component

- 1 Select **Deployment** in the Category pane.
- 2 Double click on **GetRecords** in the Instance pane.
- 3 Click **JSP** under Resources in the Category pane.
- 4 Click **PatientResponseJSP** in the Instance pane.
- 5 Drag the **PatientResponseJSP** from the Instance pane and drop it on JSP under Service Trigger in the Deployment component



- 6 Select **File>Save** from the menu.

The PatientResponseJSP Properties sheet is displayed. The URL for the JSP is PatientResponseJSP. Leave the remaining fields blank for this exercise.

EXERCISE 8-3: Deploy the Web Service with the JSP

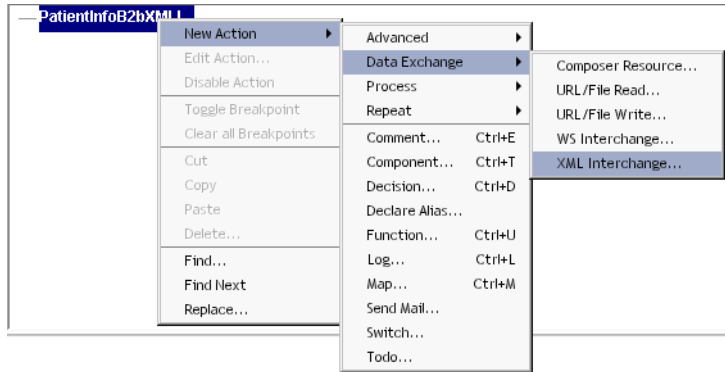
The steps to deploy your service are identical to the steps described in EXERCISE 5-5: “Deploying to the JBoss application server” on page 55. When the deployment is complete, your browser will display a deployment completed message. After receiving this message, enter:

http://localhost/GetRecords/PatientResponseJSP?physician=SSpade

in the text window of your browser. You will see the output from the service displayed by the JSP.

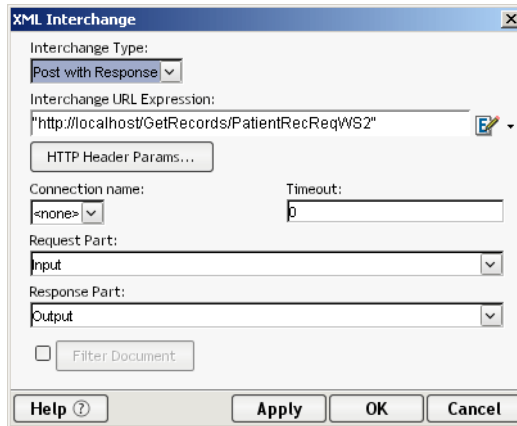
EXERCISE 8-4: Create an XML Map Component that uses XML interchange

- 1** Select **XML Map** in the Category Pane.
- 2** **RMB>New.**
The Create a New XML Map Component dialog appears.
- 3** Enter **PatientInfoB2bXMLI** in the Name textfield.
- 4** Click **Next**.
- 5** Select **patientrecords** from the Template Category dropdown list for the Input.
- 6** Select **PatientRecRequest** from the Template Name dropdown list for the Input
- 7** Select **patientrecords** from the Template Category dropdown list for the Output.
- 8** Select **PatientRecResponse** from the Template Name dropdown list for the Output.
- 9** Click **Next**.
You won't be using temp or fault documents in this component.
- 10** Click **Finish**.
- 11** Highlight the **PatientInfoB2bXMLI** component in the Action Model.
- 12** **RMB>New Action>Data Exchange>XML Interchange**



The XML Interchange Dialog appears.

- 13** Select **Post with Response** from the Interchange Type dropdown list.
- 14** Select **Input** from the Request Part dropdown list.
- 15** Select **Output** from the Response Part dropdown list.
- 16** Enter "**http://localhost/GetRecords/PatientRecReqWS2**", including quotes in the Interchange URL Expression textfield.



- 17** Click **OK**.

The interchange action is added to the component action model.



- 18** Select **File>Save** from the menu.

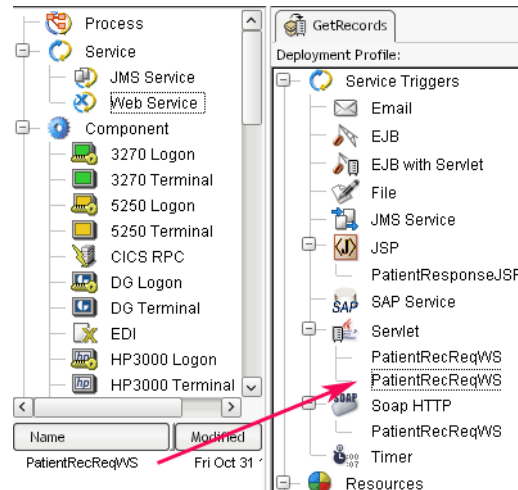
EXERCISE 8-5: Modify the Deployment Component for the XML Interchange.

The current deployment of the patient record request web service expects parameters of the type Params(URL/Form). The XML Interchange uses XML documents as its method of passing parameters. You will need to modify the deployment component. You will also create a servlet of type XML(HTTP/Post), which expects input as an XML document.

- 1 Select **Deployment** in the Category Pane.
- 2 Select **GetRecords** in the Instance Pane.

You will need to add an additional URL for the service prior to deployment. This will avoid a name collision with your previously deployed version of web service.

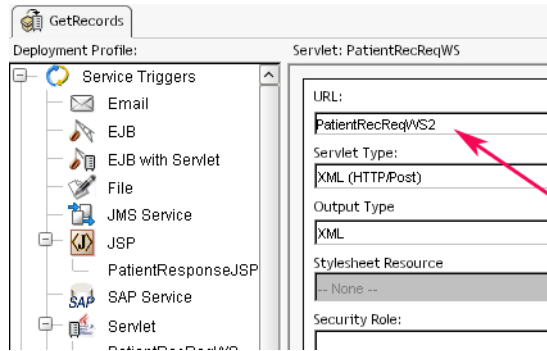
- 3 Double click on **GetRecords** in the Instance Pane, to open the component.
- 4 Select **Web Service** in the Category Pane.
- 5 Highlight **PatientRecReqWS** in the Instance Pane.
- 6 Drag and Drop a second copy of **PatientRecReqWS** onto the servlet icon under Service Triggers in the Deployment Profile.



- 7 Enter **PatientRecReqWS2** in the URL field.

The URL must be a unique name for each servlet. The XML Servlet expects an XML document as its parameters.

- 8 Select **XML(HTTP/Post)** from the Servlet Type dropdown list in the servlet properties sheet.
- 9 Select **File>Save** from the menu.

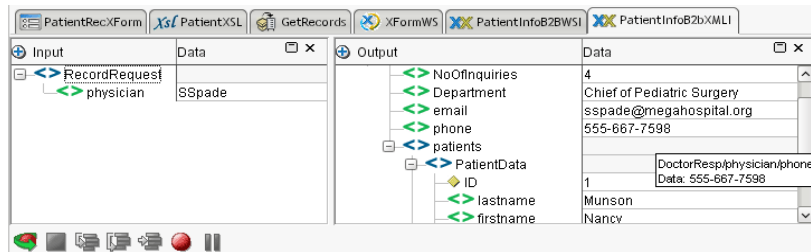


EXERCISE 8-6: Deploy the Web Service with Servlet Type XML(HTTP/Post)

Deploy the component as described in EXERCISE 8-3: “Deploy the Web Service with the JSP” on page 117. When the deployment is complete the browser will display a deployment completed message. After receiving this message, return to Integration Manager, then:

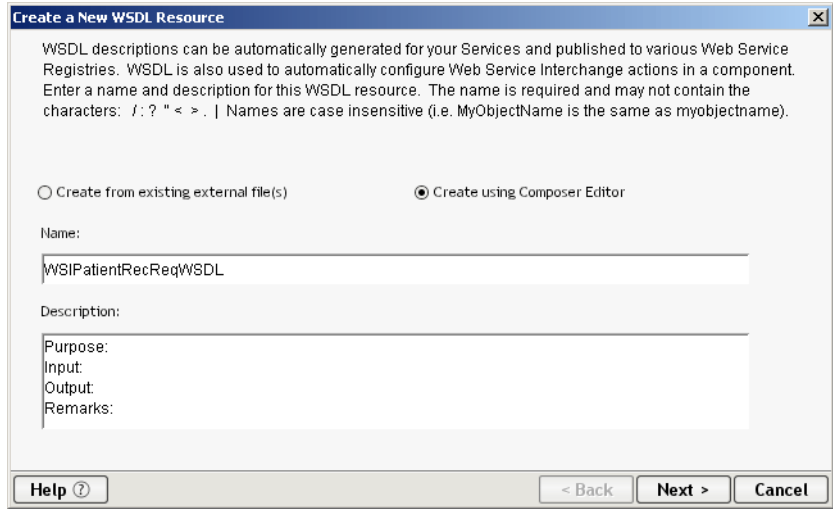
- 1 Select the **PatientInfoB2bXMLI** tab.
- 2 Select **Component>Execute** from the menu.

The service runs and returns data to the XML Map Components Output Part.

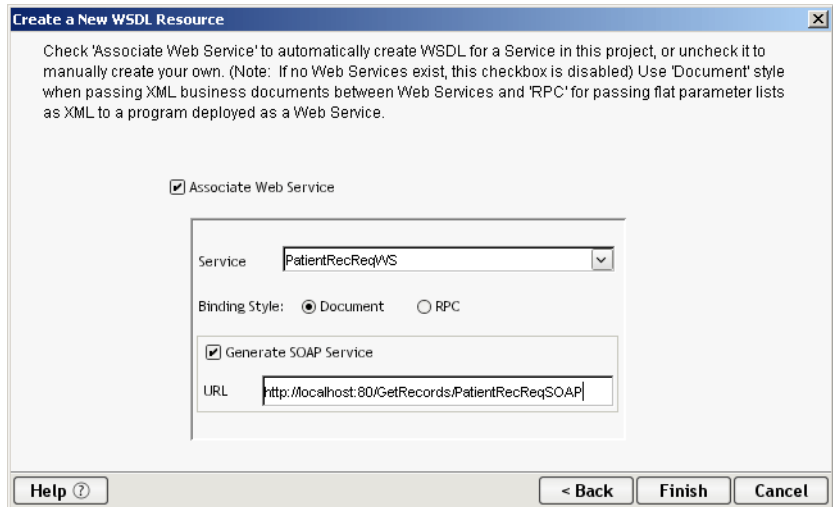


EXERCISE 8-7: Create a WSDL for the PatientRecResponse Web Service

- 1 Select **WSDL** from the Resource section of the Category Pane.
- 2 **RMB>New**.
- 3 Select the **Create using Composer Editor** radio button.
- 4 Enter **WSIPatientRecReqWSDL** in the Name textfield.

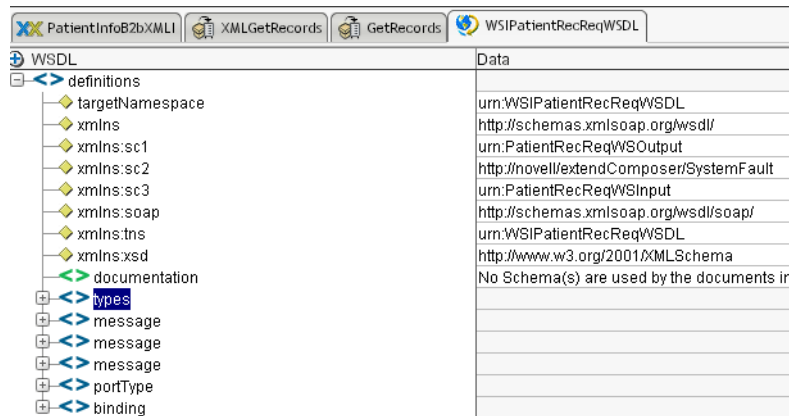


- 5 Click **Next**.
- 6 Select **PatientRecReqWS** from the Service dropdown list.
- 7 Select the **Generate SOAP Service** checkbox.
- 8 Enter **http://localhost:80/GetRecords/PatientRecReqSOAP** in the URL field.



- 9 Click **Finish**.
A message dialog appears informing you that Integration Manager has created element types of xsd:string in the WSDL.
- 10 Click **OK**.
The WSDL is created and opened in the component editor.

11 Select **File>Save** from the menu.



EXERCISE 8-8: Create XML Templates for the WSDL

You will need to have XML templates corresponding to message parts in the WSDL, in order to create working components. The templates will act as sample document that can be validated against the WSDL.

- 1** Click the **Create XML Templates** button on the Integration Manager toolbar.



- 2** Select **patientrecords** as the Template Category from the dropdown list for the Input Message.
- 3** Select **patientrecords** as the Template Category from the dropdown list for the Output Message.
- 4** Manually type **faults** in the Template Category for the Fault Message if it is not already populated by Integration Manager.

Accept the defaults, Integration Manager has created for the other fields.

Create XML Templates From WSDL

Select a Binding and Operation whose messages will be used to create XML Template Resources. Sample documents will be created for each checked message Part. Create the resource in an existing Template Category or type a new one. Accept the Template and Sample names or type new ones.

Binding:
 \WSIPatientRecReqWSDLSoapBinding

Operation:
 \WSIPatientRecReqWSDLOperation(I=tns:WSIPatientRecReqWSDLInput O=tns:WSIPatientRecReqWSDLOutput)

Input Message:
 Part(s) for Document style message: WSIPatientRecReqWSDLInput

Create	Part Name	Template Category	Template Name	Sample
<input checked="" type="checkbox"/>	RecordRequest	patientrecords	RecordRequest	RecordRequest

Output Message:
 Part(s) for Document style message: WSIPatientRecReqWSDLOutput

Create	Part Name	Template Category	Template Name	Sample
<input checked="" type="checkbox"/>	DoctorResp	patientrecords	DoctorResp	DoctorResp

Fault Message:
 Part(s) for Document style message: WSIPatientRecReqWSDLFault

Create	Part Name	Template Category	Template Name	Sample
<input checked="" type="checkbox"/>	FaultInfo	faults	FaultInfo	FaultInfo

Help ? OK Cancel

5 Click **OK**.

Integration Manager will return a message dialog “XML templates created.”

6 Click **OK**.

7 Select **File>Save ALL** from the menu.

If you look in the patientrecords folder under XML Template Categories you will see the newly created WSDL templates.

A new category “faults” has also been created, which contains the fault template.

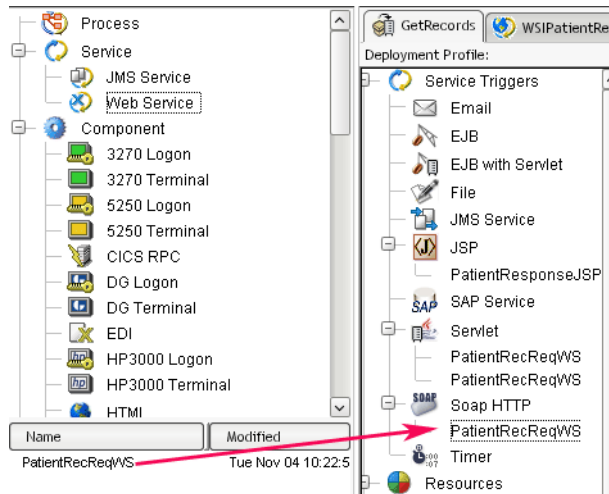
EXERCISE 8-9: Add a Soap Trigger to the Deployment Component

1 Select **Deployment** in the Category Pane.

2 Double click on **GetRecords** in the Instance Pane.

3 Select **Web Service** in the Category Pane.

4 Drag the **PatientRecReqWS** from the Instance Pane and drop it on the SOAP HTTP trigger in the Deployment Profile pane.



The properties sheet for the SOAP HTTP:PatientRecReqWS appears.

You need to modify the URL so that you won't create a name collision with the servlet triggered version of the web service.

- 5 Enter **PatientRecReqSOAP** in the URL field in the properties sheet.



- 6 Select **File>Save** from the menu to save your work.

EXERCISE 8-10: Deploy the Web Service with SOAP

Deploy the component as described in EXERCISE 8-3: “Deploy the Web Service with the JSP” on page 117. When the deployment is complete the browser will display a deployment completed message.

EXERCISE 8-11: Create an XML Map component that executes a WSInterchange action

You will create an XML Map component that uses the templates that you created from the WSDL in exercise 8-8.

- 1 Select the **XML Map Component** in the Category Pane.
- 2 **RMB>New**
- 3 Enter **PatientInfoB2BWSI** in the Name field
- 4 Click **Next**.

- 5 Select **patientrecords** as the Template Category for *both* the Input and Output parts.
- 6 Select **RecordRequest** as the Input Template Name.
- 7 Select **DoctorResp** as the Output Template Name.

Specify one or more XML Templates to help design Input to this Component or Web Service and only one to design Output. The sample XML Documents in each Template are design time aids to help you build Action Models for the component. The samples are not actually used at runtime after deployment to your application server. The Identifier is fixed and represents the name used to refer to the XML Document during component execution. Selecting System {ANY} allows you to use an empty template (i.e. accept any document as Input).

Input Message

Part	Template Category	Template Name
Input	patientrecords	RecordRequest

Output Message

Part	Template Category	Template Name
Output	patientrecords	DoctorResp

- 8 Click **Next**.
- 9 Click the **Add** button in the Fault Message section of the dialog.
- 10 Select **faults** as the Template Category for the Fault Message.
- 11 Select **FaultInfo** as the Fault Template Name.

Specify one or more Temp and Fault XML Templates to help design temporary parts and fault handling for this Component or Web Service. Use Temp documents for creating intermediate results or holding values for reference. Specify XML Templates to serve as Fault documents to be passed back to clients under error conditions.

Temp Message

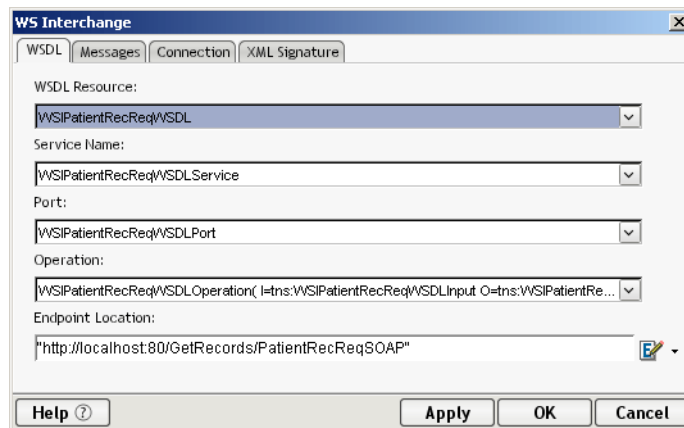
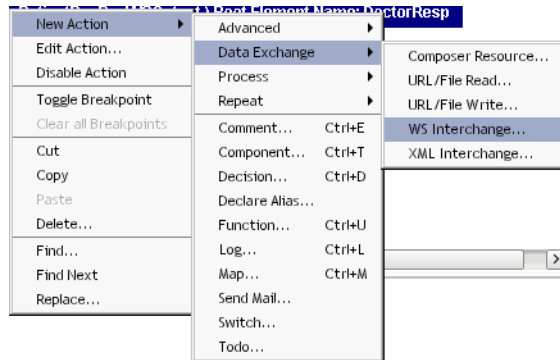
Part	Template Category	Template Name
------	-------------------	---------------

Fault Message

Part	Template Category	Template Name
_SystemFault	{System}	{Fault}
Fault	faults	FaultInfo

- 12 Click **Finish**.

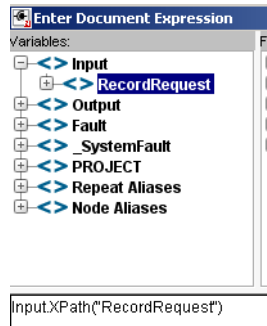
- 13** Click on the **Apply NameSpaces** statement in the Action Model.
- 14** **RMB>New Action>Data Exchange>WS Interchange.**



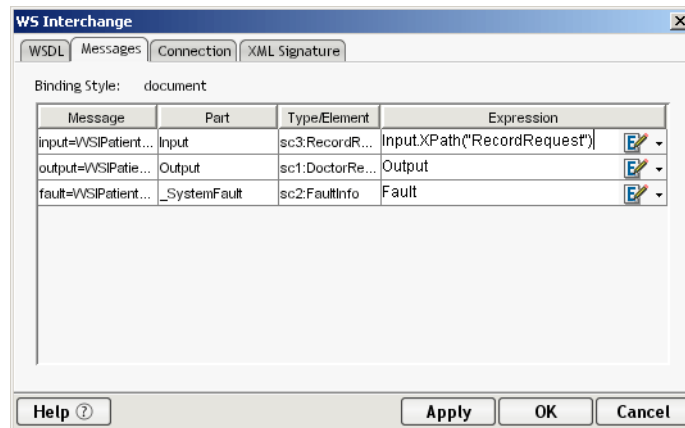
The WS Interchange dialog is populated by Integration Manager, except these value.

- 15** Click the **Messages** tab.
- 16** Click the **Expression Editor** button in the Expression list textfield for the input.
- 17** Expand the input node in the Variables pane and double click on the **RecordRequest** element.

The XPath expression appears in the editor pane.



- 18 Click **OK**.
- 19 Enter **Output** in the Expression list textfield for output.
- 20 Enter **Fault** in the Expression list textfield for fault.



- 21 Click **OK**.
- 22 Select **File>Save** from the menu.
- 23 Double click on the **data** field of the **physician** element in the input.
- 24 Enter **SSpade** in the textfield of the Edit Data dialog.
- 25 Click **OK**.
- 26 Select **Component>Execute** from the menu.
 The component executes invoking the web service via the WSInterchange action.
 You should see data in your Output part.

EXERCISE 8-12: Create an XForm for Physician Input

Integration Manager will build an XML Form for you based on the instance data in the templates you specify.

- 1 Select the **Form** in the Category Pane.
- 2 **RMB>New**
- 3 Enter **PatientRecXForm** in the Name field.

Create a New Form Resource [X]

Form Resources allow you to build a browser based User Interface served up by a Composer Web Service using the Form Process action. The action will process the Form on the server and send the resultant page to a browser. Enter a name and, optionally, a description for this Form Resource. The name may not contain the characters: / : ? " < > . | Names are case insensitive (i.e. MyObjectName is the same as myobjectname).

Name:
PatientRecXForm

Description:
Purpose:
Input:
Output:
Remarks:

Help [?] < Back Next > Cancel

- 4 Click **Next**.
- 5 Select **patientrecords** as the XML Template Category from the dropdown list.
- 6 Select **RecordRequest** as the XML Template Name.
- 7 Select **RecordRequest.xml** as the Sample Name.

Create a New Form Resource [X]

Select an XML file upon which the instance data for the Form will be built. A default Form will automatically be built upon this instance data using this file. This Form may then be further modified and edited. Select any Sample document from an existing XML Template in the current project or select a file from outside the project using URL / Local File. Then, specify a URL to handle the data submitted by the Form.

Instance Data Source Location

XML Template

Template Category	Template Name	Sample Name
patientrecords	RecordRequest	RecordRequest.xml

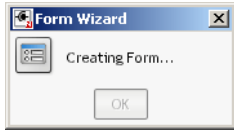
URL / Local File

Browse...

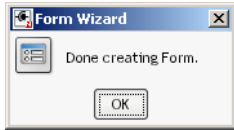
URL Target for Form Submission (Post):

Help [?] < Back Finish Cancel

- 8 Click **Finish**.
- A “Creating Form” message dialog appears.

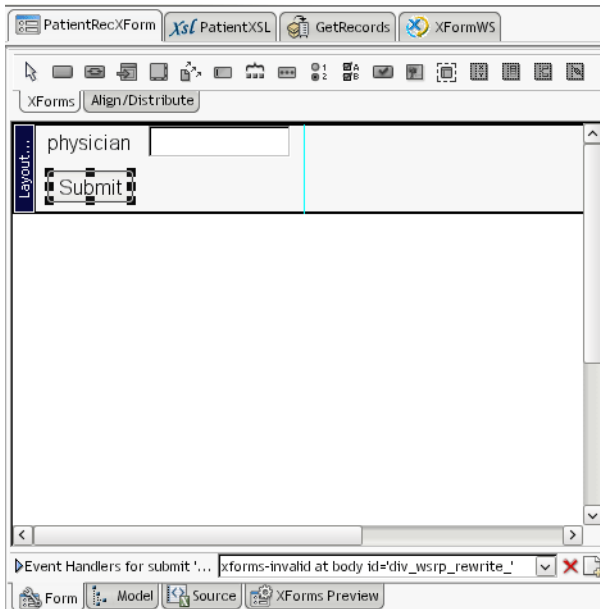


When Integration Manager has completed creating the form, the message “Done creating Form” is displayed.



9 Click **OK**.

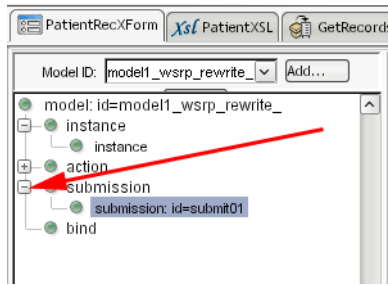
The XForm Component editor opens with the Form layout editor displayed. Note the four tabs at the bottom of the editor window.



Next you will set the URL of the service that will be invoked when the submit button is clicked.

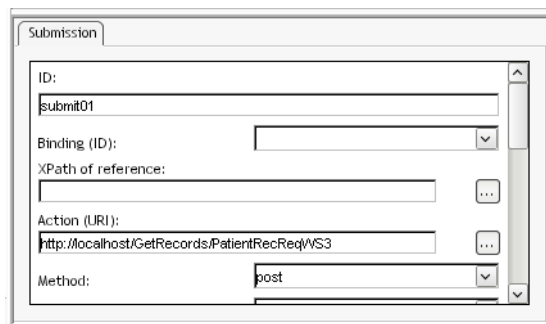
10 Select the **Model** tab in the component editor.

11 Expand the **submission** node in the editor.



12 Click on **submission id=submit01**.

The Submission property tab appears in the lower right corner of the editor. You will set the URL of the web service, and the method of passing parameters to the service in this tab.



13 Enter **http://localhost/GetRecords/PatientRecReqWS3** in the Action URI field.

14 Select **post** from the Method dropdown list.

15 Select **File>Save** from the menu.

EXERCISE 8-13: Create a Web Service that uses the XForm

In this exercise you create a web service that will process the XForm.

1 Select **Web Service** in the Category Pane.

2 **RMB>New**.

3 Enter **XFormWS** in the Name field.

4 Click **Next**.

You can accept the default templates for the templates, the XForm contains the information about the templates. You will not be using the fault or temp parts for this web service or headers. Accept the defaults for these panels.

5 Click **Next**.

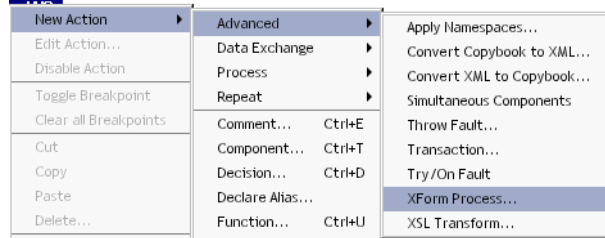
6 Click **Next**.

7 Click **Finish**.

The web service component editor appears. You will add an XForm Process action, that will use the form you created to process the input and output the result as html.

8 Highlight **XFormWS** in the component editor.

9 **RMB>New Action>Advanced>XFormProcess**.



The Form Process dialog appears.

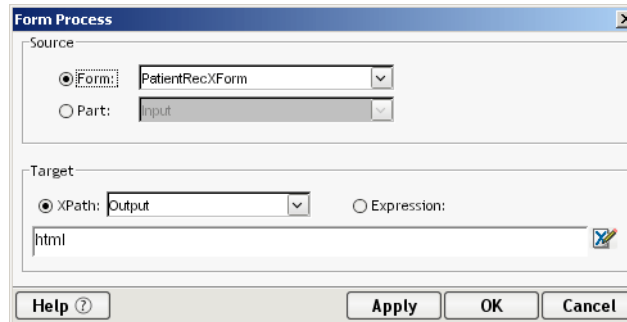
10 Select the **Form** radio button for the source.

11 Select **PatientRecXForm** from the source dropdown list.

12 Select the **XPath** radio button for the target.

13 Enter **html** in the target text box.

14 Click **OK**.



The XForm process action is added to the model.



15 Select **File>Save** from the menu.

EXERCISE 8-14: Create the deployment profile for the XFormWS

In this exercise you will deploy both the XFormWS you just created, and the PatientRecReqWS. The PatientRecReqWS will be deployed as a servlet, which expects XHTML output by the XForm. You will also import a XSL style sheet that will format the output from the PatientRecReqWS in the browser.

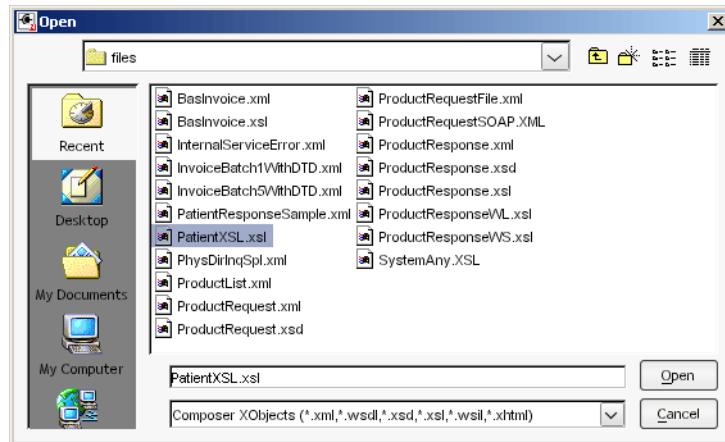
- 1 Select **XSL** in the Category Pane.
- 2 **RMB>Import XObject.**



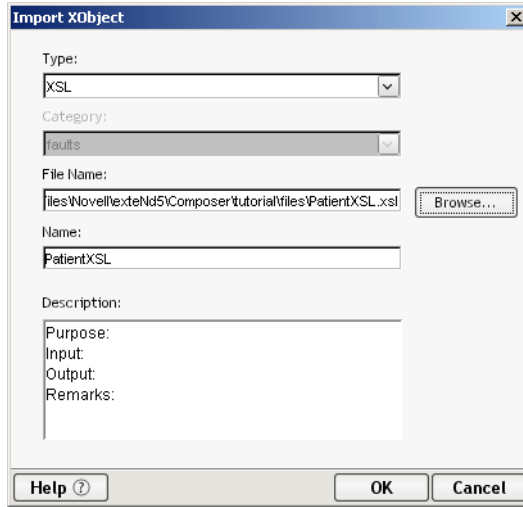
The Import XObject dialog appears.

- 3 Click the **Browse** button on the dialog.
- 4 Navigate to the `..\tutorial\files` directory.

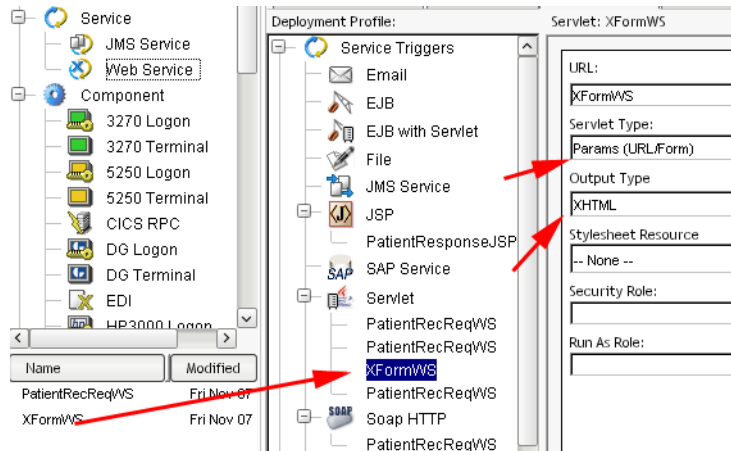
This directory is located under where you installed Integration Manager.



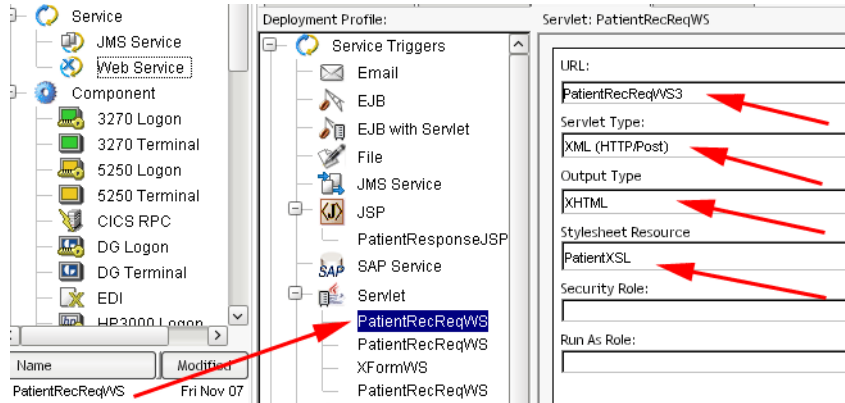
- 5 Double click the file **PatientXSL.xml** to select it.



- 6 Click **OK**.
- PatientXSL is added to the Instance Pane.
- 7 Click on **Deployment** in the Category Pane.
- 8 Double click on the **GetRecords** deployment object in the Instance Pane to open it.
- 9 Select **Web Service** in the Category Pane.
- 10 Drag the **XFormWS** from the Instance Pane and drop it on the **Servlet** section of the Service Triggers in the Deployment Profile pane.
- 11 Select **Params(URL/Form)** as the Servlet Type from the dropdown.
- 12 Select **XHTML** as the Output Type.



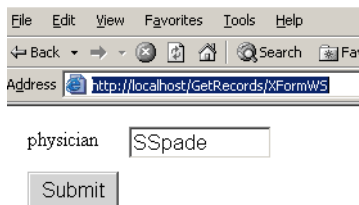
- 13** Drag **PatientRecReqWS** from the Instance Pane and drop it onto the **Servlet** section of the Service Triggers in the Deployment Profile pane.
- 14** Enter **PatientRecReqWS3** in the URL field.
- 15** Select **XML (HTTP/Post)** as the Servlet Type from the dropdown list.
- 16** Select **XHTML** as the Output Type from the dropdown list.
- 17** Select **PatientXSL** from the dropdown list as the Stylesheet Resource.
- 18** Select **File>Save** from the menu.



EXERCISE 8-15: Deploy the XForm Web Service

Deploy the component as described in EXERCISE 8-3: “Deploy the Web Service with the JSP” on page 117. When the deployment is complete the browser will display a deployment completed message. After receiving this message, return to Integration Manager, then:

- 1** Enter **http://localhost/GetRecords/XFormWS** in your browser URL.
The XForm displays with physician field populated.



- 2** Click the **Submit** button in the browser window.
- 3** The data is displayed in the browser window.

Summary of what you've done

- ◆ Created a JSP resource
- ◆ Deployed the PatientRecReqWS with a servlet trigger using a JSP
- ◆ Utilized XML Interchange as a consumer of the web service
- ◆ Created a WSDL for the PatientRecReqWS
- ◆ Deployed the PatientRecReqWS with a SOAP HTTP service trigger
- ◆ Utilized WSInterchange to consume the web service.
- ◆ Created an XForm.
- ◆ Deployed the XForm in a web service that processes the input and invokes the PatientRecReq web service.

What's next

Congratulations. You've completed the Novell Integration Manager Tutorial. To learn more about Integration Manager, consult your *Novell Integration Manager User Guide* and the system help files. In addition, Novell offers classroom training on Integration Manager, refer to the Novell website for more information.
http://www.novell.com/training/train_product/extend.html.

Index

A

Action 84
 Comment 84
 Component 85
 Decision 86
Action Model 84
architecture, Integration Manager 18

C

Component 9
 JDBC 63
 LDAP 73
Component Animation 19
Composer
 environment 7
Connection 60
 JDBC 60
 LDAP 69
ConsumingWS 113
Create a Web Service 38
Creating and Importing XML Templates 30

D

Deployment 44
 Considerations 44
 Profiles 45
 Service Triggers 52
DOM 18

E

ECMAScript 86
Editor Frame 11
environment, Integration Manager 7
Expression Builder 86

I

Integration Manager
 architecture 18

J

JSP 113

L

Launch Integration Manager 10

M

Message Frame 11
MySQL
 prerequisite 5
 setup 8

N

Navigation Frame 11

P

Project Create 14
Project Structure 12
Publishing Web Services 111
PublishWS 112

R

Resource 9

S

Service 9
Service vs. Components 38
Servlets 112

T

Template 9

W

Web Service Definition 38

WSDL 112

WS Interchange 114

X

XML

Sample Documents 28

Schema 28

Template Categories 28

Templates 29

XML Interchange 114

XML Map Action 19

xObject 9

XPath 86

XSL Style Sheets 29