Informatica® B2B Data Transformation
10.2

XMap Tutorial

Informatica B2B Data Transformation XMap Tutorial
10.2
April 2017

© Copyright Informatica LLC 2013, 2018

This software and documentation are provided only under a separate license agreement containing restrictions on use and disclosure. No part of this document may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording or otherwise) without prior consent of Informatica LLC.

Informatica, the Informatica logo, and PowerCenter are trademarks or registered trademarks of Informatica LLC in the United States and many jurisdictions throughout the world. A current list of Informatica trademarks is available on the web at https://www.informatica.com/trademarks.html. Other company and product names may be trade names or trademarks of their respective owners.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/), and/or other software which is licensed under various versions of the Apache License (the "License"). You may obtain a copy of these Licenses at http://www.apache.org/licenses/. Unless required by applicable law or agreed to in writing, software distributed under these Licenses is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the Licenses for the specific language governing permissions and limitations under the Licenses.

This product includes software which was developed by Mozilla (http://www.mozilla.org/), software copyright The JBoss Group, LLC, all rights reserved; software copyright © 1999-2006 by Bruno Lowagie and Paulo Soares and other software which is licensed under various versions of the GNU Lesser General Public License Agreement, which may be found at http:// www.gnu.org/licenses/lgpl.html. The materials are provided free of charge by Informatica, "as-is", without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

The product includes ACE(TM) and TAO(TM) software copyrighted by Douglas C. Schmidt and his research group at Washington University, University of California, Irvine, and Vanderbilt University, Copyright (©) 1993-2006, all rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (copyright The OpenSSL Project. All Rights Reserved) and redistribution of this software is subject to terms available at http://www.openssl.org and http://www.openssl.org/source/license.html.

This product includes Curl software which is Copyright 1996-2013, Daniel Stenberg, <daniel@haxx.se>. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://curl.haxx.se/docs/copyright.html. Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

The product includes software copyright 2001-2005 (©) MetaStuff, Ltd. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://www.dom4j.org/ license.html.

The product includes software copyright © 2004-2007, The Dojo Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http://dojotoolkit.org/license.

This product includes ICU software which is copyright International Business Machines Corporation and others. All rights reserved. Permissions and limitations regarding this software are subject to terms available at http://source.icu-project.org/repos/icu/icu/trunk/license.html.

This product includes software copyright © 1996-2006 Per Bothner. All rights reserved. Your right to use such materials is set forth in the license which may be found at http:// www.gnu.org/software/ kawa/Software-License.html.

This product includes OSSP UUID software which is Copyright © 2002 Ralf S. Engelschall, Copyright © 2002 The OSSP Project Copyright © 2002 Cable & Wireless Deutschland. Permissions and limitations regarding this software are subject to terms available at http://www.opensource.org/licenses/mit-license.php.

This product includes software developed by Boost (http://www.boost.org/) or under the Boost software license. Permissions and limitations regarding this software are subject to terms available at http:/ /www.boost.org/LICENSE_1_0.txt.

This product includes software copyright © 1997-2007 University of Cambridge. Permissions and limitations regarding this software are subject to terms available at http:// www.pcre.org/license.txt.

This product includes software copyright © 2007 The Eclipse Foundation. All Rights Reserved. Permissions and limitations regarding this software are subject to terms available at http:// www.eclipse.org/org/documents/epl-v10.php and at http://www.eclipse.org/org/documents/edl-v10.php.

This product includes software licensed under the terms at http://www.tcl.tk/software/tcltk/license.html, http://www.bosrup.com/web/overlib/?License, http:// www.stlport.org/doc/ license.html, http://asm.ow2.org/license.html, http://www.cryptix.org/LICENSE.TXT, http://hsqldb.org/web/hsqlLicense.html, http:// httpunit.sourceforge.net/doc/ license.html, http://jung.sourceforge.net/license.txt , http://www.gzip.org/zlib/zlib_license.html, http://www.openldap.org/software/ release/license.html, http://www.libssh2.org, http://slf4j.org/license.html, http://www.sente.ch/software/OpenSourceLicense.html, http://fusesource.com/downloads/ license-agreements/fuse-message-broker-v-5-3- license-agreement; http://antlr.org/license.html; http://aopalliance.sourceforge.net/; http://www.bouncycastle.org/ licence.html; http://www.jgraph.com/jgraphdownload.html; http://www.jcraft.com/jsch/LICENSE.txt; http://jotm.objectweb.org/bsd_license.html; . http://www.w3.org/ Consortium/Legal/2002/copyright-software-20021231; http://www.slf4j.org/license.html; http://nanoxml.sourceforge.net/orig/copyright.html; http://www.json.org/ license.html; http://forge.ow2.org/projects/javaservice/, http://www.postgresql.org/about/licence.html, http://www.sqlite.org/copyright.html, http://www.tcl.tk/ software/tcltk/license.html, http://www.jaxen.org/faq.html, http://www.jdom.org/docs/faq.html, http://www.slf4j.org/license.html; http://www.iodbc.org/dataspace/ iodbc/wiki/iODBC/License; http://www.keplerproject.org/md5/license.html; http://www.toedter.com/en/jcalendar/license.html; http://www.edankert.com/bounce/ index.html; http://www.net-snmp.org/about/license.html; http://www.openmdx.org/#FAQ; http://www.php.net/license/3_01.txt; http://srp.stanford.edu/license.txt; http://www.schneier.com/blowfish.html; http://www.jmock.org/license.html; http://xsom.java.net; http://benalman.com/about/license/; https://github.com/CreateJS/ EaselJS/blob/master/src/easeljs/display/Bitmap.js; http://www.h2database.com/html/license.html#summary; http://jsoncpp.sourceforge.net/LICENSE; http:// jdbc.postgresql.org/license.html; http://protobuf.googlecode.com/svn/trunk/src/google/protobuf/descriptor.proto; https://github.com/rantav/hector/blob/master/ LICENSE; http://web.mit.edu/Kerberos/krb5-current/doc/mitK5license.html; http://jibx.sourceforge.net/jibx-license.html; https://github.com/lyokato/libgeohash/blob/ master/LICENSE; https://github.com/hjiang/jsonxx/blob/master/LICENSE; https://code.google.com/p/lz4/; https://github.com/jedisct1/libsodium/blob/master/ LICENSE; http://one-jar.sourceforge.net/index.php?page=documents&file=license; https://github.com/EsotericSoftware/kryo/blob/master/license.txt; http://www.scala-lang.org/license.html; https://github.com/tinkerpop/blueprints/blob/master/LICENSE.txt; http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/ intro.html; https://aws.amazon.com/asl/; https://github.com/twbs/bootstrap/blob/master/LICENSE; https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/ LICENSE.txt; https://github.com/documentcloud/underscore-contrib/blob/master/LICENSE, and https://github.com/apache/hbase/blob/master/LICENSE.txt.

This product includes software licensed under the Academic Free License (http://www.opensource.org/licenses/afl-3.0.php), the Common Development and Distribution License (http://www.opensource.org/licenses/cddl1.php) the Common Public License (http://www.opensource.org/licenses/cpl1.0.php), the Sun Binary Code License Agreement Supplemental License Terms, the BSD License (http:// www.opensource.org/licenses/bsd-license.php), the new BSD License (http:// opensource.org/licenses/BSD-3-Clause), the MIT License (http://www.opensource.org/licenses/mit-license.php), the Artistic License (http://www.opensource.org/ licenses/artistic-license-1.0) and the Initial Developer's Public License Version 1.0 (http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/).

This product includes software copyright © 2003-2006 Joe Walnes, 2006-2007 XStream Committers. All rights reserved. Permissions and limitations regarding this software are subject to terms available at http://xstream.codehaus.org/license.html. This product includes software developed by the Indiana University Extreme! Lab. For further information please visit http://www.extreme.indiana.edu/.

This product includes software Copyright (c) 2013 Frank Balluffi and Markus Moeller. All rights reserved. Permissions and limitations regarding this software are subject to terms of the MIT license.

See patents at https://www.informatica.com/legal/patents.html.

DISCLAIMER: Informatica LLC provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of noninfringement, merchantability, or use for a particular purpose. Informatica LLC does not warrant that this software or documentation is error free. The information provided in this software or documentation may include technical inaccuracies or typographical errors. The information in this software and documentation is subject to change at any time without notice.

NOTICES

This Informatica product (the "Software") includes certain drivers (the "DataDirect Drivers") from DataDirect Technologies, an operating company of Progress Software Corporation ("DataDirect") which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

The information in this documentation is subject to change without notice. If you find any problems in this documentation, please report them to us in writing at Informatica LLC 2100 Seaport Blvd. Redwood City, CA 94063.

INFORMATICA LLC PROVIDES THE INFORMATION IN THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT.

Publication Date: 2018-06-09

Table of Contents

Preface 6

Informatica Resources. 6

Informatica Network. 6

Informatica Knowledge Base. 6

Informatica Documentation. 7

Informatica Product Availability Matrixes. 7

Informatica Velocity. 7

Informatica Marketplace. 7

Informatica Global Customer Support. 7

Chapter 1: Introduction to XMap 8

XMap Overview. 8

The XMap Editor. 9

Mapping Statements. 9

Mapping Statement Types. 10

Scenario. 11

Chapter 2: Creating an XMap 12

Creating an XMap Overview. 12

Step 1. Create the Data Processor Transformation. 13

Step 2. Create an XMap in the Data Processor Transformation. 13

Creating XMap - Tips. 14

Chapter 3: Configuring Repeating Values 15

Configuring Repeating Values Overview. 15

Step 1. Create a Repeating Group Statement. 16

Step 2. Test the Transformation. 17

Testing XMap - Tips. 18

Chapter 4: Configuring XPath Expressions 20

Configuring XPath Expressions Overview. 20

Step 1. Create an XPath Function Expression. 21

Step 2. Test the Transformation. 23

Writing XPath Expression - Tips. 24

Chapter 5: Configuring Group Statements 25

Configuring Group Statements Overview. 25

Step 1. Create a Group Statement with Conditions. 26

Step 2. Test the Transformation. 29

Chapter 6: Configuring Router Statements. 31

Configuring Router Statements Overview. 31

Step 1. Create a Router Statement. 32

Step 2. Test the Transformation. 34

Appendix A: Frequently Asked Questions. 35

Frequently Asked Questions. 35

Appendix B: Glossary. 36

Preface

The Data Transformation *XMap Getting Started Guide* is written for developers. It provides a tutorial to help first-time users learn how to use the Data Processor transformation and XMap. This guide assumes that you have an understanding of XML source files, XML schemas, XPath expressions, and the Informatica Developer.

Informatica Resources

Informatica Network

Informatica Network hosts Informatica Global Customer Support, the Informatica Knowledge Base, and other product resources. To access Informatica Network, visit https://network.informatica.com.

As a member, you can:

- Access all of your Informatica resources in one place.
- Search the Knowledge Base for product resources, including documentation, FAQs, and best practices.
- View product availability information.
- Review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

As a member, you can:

- Access all of your Informatica resources in one place.
- Search the Knowledge Base for product resources, including documentation, FAQs, and best practices.
- View product availability information.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to search Informatica Network for product resources such as documentation, how-to articles, best practices, and PAMs.

To access the Knowledge Base, visit https://kb.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

To get the latest documentation for your product, browse the Informatica Knowledge Base at https://kb.informatica.com/_layouts/ProductDocumentation/Page/ProductDocumentSearch.aspx.

If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com.

Informatica Product Availability Matrixes

Product Availability Matrixes (PAMs) indicate the versions of operating systems, databases, and other types of data sources and targets that a product release supports. If you are an Informatica Network member, you can access PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions.

If you are an Informatica Network member, you can access Informatica Velocity resources at http://velocity.informatica.com.

If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that augment, extend, or enhance your Informatica implementations. By leveraging any of the hundreds of solutions from Informatica developers and partners, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through Online Support on Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link: http://www.informatica.com/us/services-and-training/support-services/global-support-centers.

If you are an Informatica Network member, you can use Online Support at http://network.informatica.com.

CHAPTER 1

Introduction to XMap

This chapter includes the following topics:

- XMap Overview, 8
- The XMap Editor, 9
- Mapping Statements, 9
- Mapping Statement Types, 10
- Scenario, 11

XMap Overview

A mapping uses a Data Processor transformation to transform documents from one format to another. An XMap is a Data Processor transformation object that transforms an XML input source to another XML with a different hierarchy structure. Use the XMap editor grid in the Developer to define the XMap.

The Data Processor transformation processes unstructured and semi-structured file formats in a mapping. When you create a Data Processor transformation, you define components, to transform the data. A Data Processor transformation can contain multiple components to process data. Each component might contain other components.

After you create a Data Processor transformation, you add an XMap to transform the data. You configure the XMap with input and output schemas and an example source. An XMap uses input and output schemas to define the expected hierarchy of input and output documents. An XMap can transform any input XML document whose elements match the input schema hierarchy into an output document with the hierarchy of the output schema.

If a Data Processor transformation contains multiple XMaps, you need schemas for each component that you define in the transformation. The Data Processor transformation references schema objects in the Model repository. The schema objects can exist in the repository before you create the transformation.

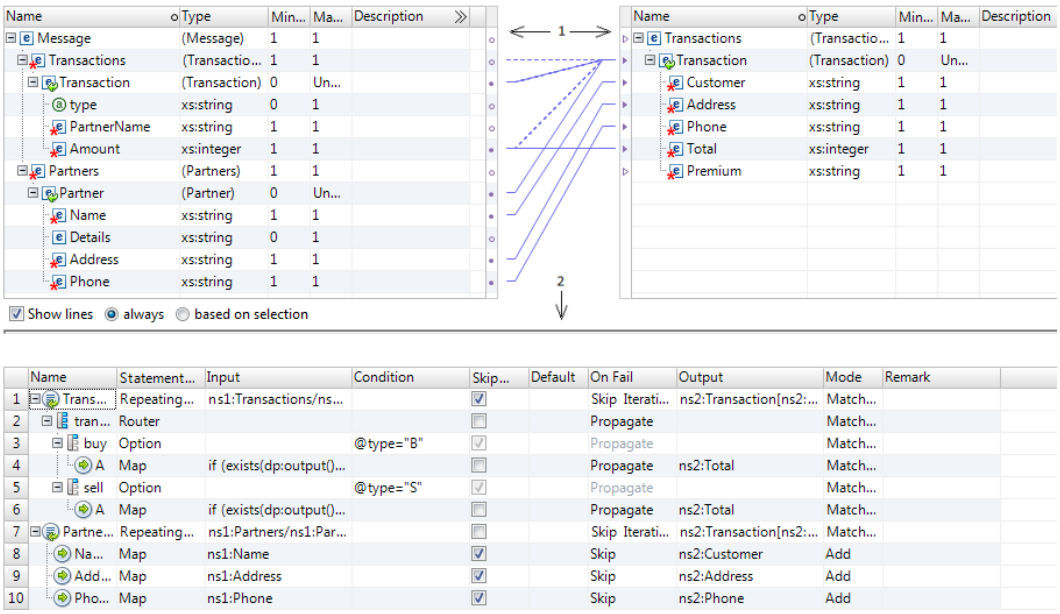
After you create an XMap, use the XMap editor to define and manage mapping statements that the transformation uses to transform a specific input element into a specific output element. The XMap editor contains the input schema hierarchy and the output XML schema hierarchy. Mapping statements link input schema elements to output schema elements.

The XMap Editor

An XMap uses input and output schemas to define the expected hierarchy of the input and output XML. Use the XMap editor to define how input elements are mapped to output elements.

An XMap can transform any input XML document whose elements match the input schema hierarchy into an output document with the hierarchy of the output schema.

The following image shows the XMap editor:



1. The XMap editor contains input and output XML schemas. Drag and drop between the schema elements to create mapping statements.
2. The XMap editor grid shows mapping statements. Use the grid to manage and edit the mapping statements.

An XMap uses mapping statements to define how to transform an input schema element to an output schema element. You can drag from a node in the input schema to a node in the output schema to create a link. When you create links, these are mapping statements.

The XMap editor shows the mapping statement in the grid. You can edit the mapping statements in the grid.

Mapping Statements

A mapping statement determines how to map data from the input XML document to the output XML document. When you drag a node from the input schema to the output schema, the XMap editor creates mapping statements in the grid.

You can use the grid to create simple or detailed mapping statements. You can drag elements from the input or output schemas into fields in the grid to include them in mapping statements.

You can check the element that a mapping statement references. When you click a mapping statement in the grid, the XMap editor highlights the nodes in the schemas.

Add XPath expressions to determine the context or add computations to a mapping statement. When an XPath expression identifies the context for a mapping statement, the Data Processor transformation runs the mapping statement for each occurrence of the input element or expression in the input document.

Nest mapping statements to make them dependent on other mapping statements. A mapping statement can be a parent to a group of child statements. Each time that the Data Processor transformation runs the parent statement, it runs the child statements also. Child statements appear indented from the parent in the XMap editor.

Mapping Statement Types

Mapping statement types define XMap mapping logic. Define the mapping statement type based on whether you want to map a simple input value to an output value, iterate over an element, or perform the mapping based on a condition.

Create a statement by dragging an input schema element to an output schema element, or adding a mapping statement to the grid. When you create a statement, the Data Processor transformation identifies a mapping statement type based on whether the element is a simple element, a complex element, or a repeating element.

The basic mapping statement type is a Map, which maps a simple input value to a simple output value. Other mapping statements identify conditions or alternatives for mapping logic, or group a set of logical statements.

You can define the following types of mapping statements in the grid:

Map

Maps a simple input element to a simple output element. A Map statement is the basic building block of the XMap.

Group

A logical group of statements. Other mapping statement types are nested under the Group statement.

Repeating Group

A group statement that the Data Processor transformation performs each time the input element appears in the input document. The Repeating Group contains Map statements which are iterated. The Repeating Group identifies the element used to iterate the group.

Router

Contains a group of Option statements, and selects only the Option statement whose condition criteria matches the input. If none of the Options apply and there is a Default statement, the Default action is taken. If none of the Options apply and there is no Default statement, the Router fails.

Option

One or more Option statements are nested under the Router statement. The Option statement is like a Group statement, and contains a logical group of statements. The Option statement defines a condition to map the input element to the output element.

Default

One Default statement can be nested under the Router statement. The Default statement is performed when none of the Option statements apply. If the all the Option statements fail and there is no Default statement, the Router fails.

Run XMap

Calls another XMap object in the Data Processor transformation.

Mapping statements contain fields that you can configure to customize the statement. You can configure the input, output, and condition for mapping an input element to an output element.

Configure whether to skip a mapping statement when it fails or there is no input. Configure whether the Data Processor transformation adds an output element or matches an existing element with a value from a mapping statement.

Scenario

Verity Outpatient Clinic uses an online tracking system to process insurance claim data for doctors. The clinic uses doctor and patient data and appointment information to check if information is missing for claims.

They need to create an XMap that transforms logs with contact information and appointment information into sets of data that is relevant for their insurance processing system. The online system stores logs in XML format. They use a Data Processor transformation with an XMap that inputs doctor and patient data, sorts the data relevant to each patient insurance claim, and outputs patient insurance data sorted from claims that are missing information.

The XMap is configured with the following objects:

PatientInputSchema

The schema that defines the XML hierarchy of data for input logs.

PatientOutputSchema

The schema that defines the XML hierarchy of data for output data.

PatientCareListExample

An example source file with sample data that you use to test the XMap.

The XMap uses the PatientInputSchema schema to define and classify the data in the input log, and the PatientOutputSchema schema to define and classify the data in the output log. The XMap processes and transforms the data in the input, then stores the output in a target XML.

Before you begin the tutorials to create an XMap for Verity Outpatient Clinic, download the Patient_DP.zip file from the following link: https://mysupport.informatica.com/docs/DOC-9857

Add the file to the following directories:

<INSTALL_DIR>\clients\DeveloperClient\Tutorials

<INSTALL_DIR>\server\Tutorials

Unzip the files to access the input schema file, the output schema file, and example source file. The zip also contains the solution as a Metadata file, named `XMap_Getting_Started-Solution.xml`

CHAPTER 2

Creating an XMap

This chapter includes the following topics:

- Creating an XMap Overview, 12
- Step 1. Create the Data Processor Transformation, 13
- Step 2. Create an XMap in the Data Processor Transformation, 13
- Creating XMap - Tips, 14

Creating an XMap Overview

In this lesson, you create a Data Processor transformation using the Developer tool. After you create a Data Processor transformation, you add an XMap object. To set up the Developer tool, you connect to the Model repository and create a project and folder to store your work. You select the default Data Integration Service to preview data and run mappings.

Lesson Concepts

A mapping uses a Data Processor transformation to transform documents from one format to another. The Data Processor transformation processes unstructured and semi-structured file formats in a mapping. When you create a Data Processor transformation, you define components such as the XMap to transform the data. A Data Processor transformation can also contain components that perform other transformation tasks.

After you create a Data Processor transformation, you add an XMap to transform the data. You configure the XMap with input and output schemas and an example source. An XMap uses input and output schemas to define the expected hierarchy of input and output documents. An XMap can transform any input XML document whose elements match the input schema hierarchy into an output document with the hierarchy of the output schema.

Lesson Objectives

In this lesson, you complete the following tasks:

- Create a Data Processor transformation to contain components that transform documents.
- Create an XMap in the Data Processor transformation to transform input with an XML hierarchy to output with a different XML hierarchy.

Lesson Prerequisites

Before you start this lesson, verify the following prerequisites:

- Informatica Data Transformation is installed, and the services are running.
- You have configured the Developer tool.

- You have downloaded the files for the tutorial. Download the Patient_DP.zip file from the following link: https://kb.informatica.com/proddocs/Product Documentation/3/Patient_DP.zip

  Add the file to the `<INSTALL_DIR>\clients\DeveloperClient\Tutorials` directory. Unzip the file to access the input schema file, output schema file, and example source file.

Lesson Timing

Set aside 5 to 10 minutes to complete the tasks in this lesson.

Step 1. Create the Data Processor Transformation

Create the Data Processor transformation in the Model repository.

1. In the Developer tool, click **File** > **New** > **Transformation**.
2. Select the Data Processor transformation and click **Next**.
3. Select to create a blank Data Processor transformation.
4. Enter the name Patient_DP for the transformation and browse for a Model repository location to put the transformation.
5. Click **Finish**.

   The Developer tool creates the transformation in the repository. The **Overview** view appears in the Developer tool. You can update the transformation ports after you define the schemas and objects in the transformation.

Step 2. Create an XMap in the Data Processor Transformation

Create a Data Processor transformation and then create an XMap for the transformation. When you create an XMap, you must have a schema that describes the input and the output XML documents. You select the element in the schema that is the root element for the input XML.

1. In the Developer Data Processor transformation **Objects** view, click **New.**
2. Select XMap and click **Next.**
3. Enter the name **Patient_Claims** for the XMap.
4. The XMap component is the first component to process data in the transformation, so ensure that **Set as startup component** is selected.

   A Data Processor transformation can contain multiple components to process data. Each component might contain other components. You must identify which component is the entry point for the transformation.

   Click **Next.**
5. To import a schema and add it to the repository, click **Create a new schema object** and browse for the **PatientInputSchema.xsd** file in the following directory: `<INSTALL_DIR>\clients\DeveloperClient\Tutorials`

Note: To add a schema that is already in the repository, select **Add a Schema Object**, then select the schema. However, we are using a schema that is not in the repository.

6. To add the sample XML file that you use to test the XMap, browse for and select the **PatientCareListExample.xml** file in the following directory: `<INSTALL_DIR>\clients\DeveloperClient\Tutorials`
7. After you select both a schema and a sample XML file, the wizard identifies the root for the input hierarchy.

   In the **Root Element Selection** dialog box, the **in:Input** element in the schema is selected as the root element for the XML.
8. To add the schema to the repository, click **Create a new schema object** and browse for the **PatientOutputSchema.xsd** file in the following directory: `<INSTALL_DIR>\clients\DeveloperClient\Tutorials`

   Note: To add a schema that is already in the repository, select **Add a Schema Object**, then select the schema.
9. Choose the root for the output hierarchy.

   In the **Root Element Selection** dialog box, select the **out:Insurance** element in the schema as the root element for the XML.
10. Click **Finish**.

    The Developer tool creates a view for the **Patient_Claims** XMap that you created. Click the view to configure the XMap.

Creating XMap - Tips

Use the following tips to help you configure an XMap more effectively.

Set the XMap as the startup component.

A Data Processor transformation can contain multiple components to process data. Each component might contain other components. Even if there is only one component, you must identify that the component is the entry point for the transformation. If you do not select a startup component, the Data Processor transformation will not run correctly. To select a startup component, define the XMap as a startup component when you create the XMap, or select the startup component on the **Overview** tab.

Find a root element in a schema.

You can use pattern searching to search for a root element. To find a match to any number of characters in a string, enter `*<string>`. To match a single character, enter `?<character>`.

Select an example source.

You can change the example source to test the XMap at any time. You can test any example source with the same XML hierarchy as the input schema.

CHAPTER 3

Configuring Repeating Values

This chapter includes the following topics:

- Configuring Repeating Values Overview, 15
- Step 1. Create a Repeating Group Statement, 16
- Step 2. Test the Transformation, 17
- Testing XMap - Tips, 18

Configuring Repeating Values Overview

In this lesson, you configure the **Patient_Claims** XMap to pass repeating sets of details for doctors to the output.

Lesson Concepts

A mapping statement determines how to map data from the input XML document to the output XML document. When you drag a node from the input schema to the output schema, the XMap editor creates mapping statements in a grid. You can edit mapping statement fields in the grid to change the logic for a mapping statement.

A Repeating Group statement is a mapping statement that can occur multiple times. The Data Processor transformation performs the Repeating Group statement for each element or value that is a result of the Input XPath expression.

The Repeating Group input is an XPath expression that can evaluate to a sequence of elements or values. XPath is a query language used to select nodes in an XML document and perform computations.

Repeating Group statements can contain Map statements. A Map statement is the basic XMap object and maps a simple input value to a simple output value. The input is a single value or a constant value.

Lesson Objectives

In this lesson, you complete the following tasks:

- Open an XMap in the XMap editor.
- Create a Repeating Group statement to pass repeating sets of details for doctors to the output.
- Create Map statements to pass a specific doctor detail to the output.
- Validate the transformation.

Lesson Prerequisites

Before you start this lesson, verify the following prerequisites:

- You have created a Data Processor transformation.
- You have created the **Patient_Claims** XMap.

Lesson Timing

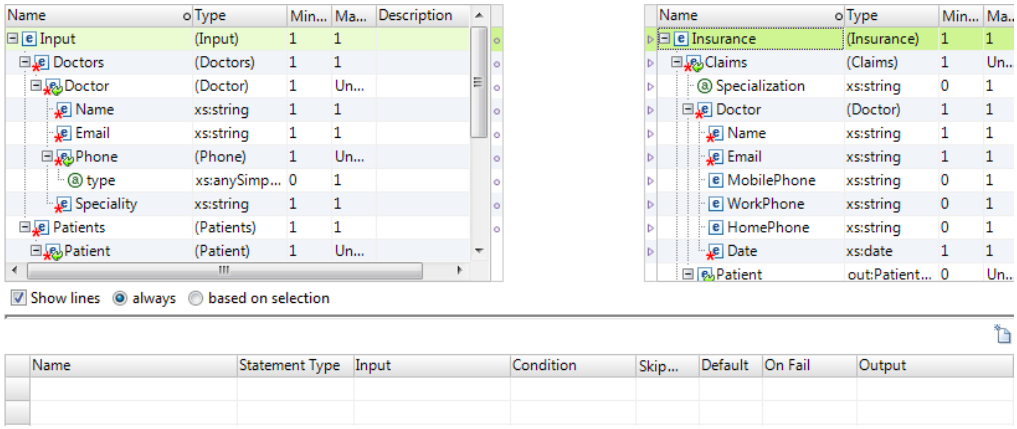
Set aside 15 to 20 minutes to complete the tasks in this lesson.

Step 1. Create a Repeating Group Statement

You can create a mapping statement by dragging a node from the input schema to a node in the output schema. The XMap editor adds the mapping statement to the mapping statement grid.

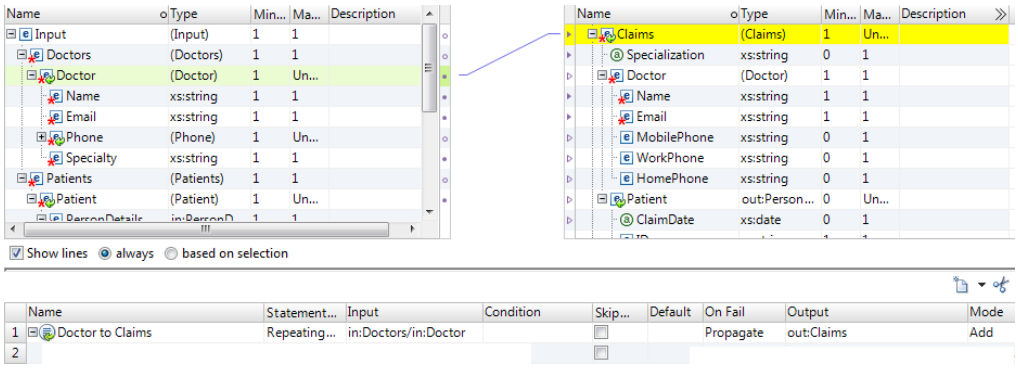
1. To open the XMap editor, in the **Overview** view, click the **Patient_Claim** XMap link.

   The XMap editor appears and shows the input and output schemas. In the following image, the XMap editor displays the input schema on the left, and the output schema on the right:



2. Drag the mouse from the **Doctor** input schema node to the **Claims** output schema node.

   The following image shows how the XMap links the XML input to the XML output:



   Both schema nodes are repeating nodes, so the XMap editor creates a Repeating Group statement named **Doctor to Claims** in the grid. Statements under the Repeating Group statement are iterated for each instance of input data.

3. Drag the mouse from the **Specialty** input node under the **Doctor** input node to the **Specialization** output schema node.

   Note: There is a **Specialty** input node under the **Doctor** input node, and a **Speciality** input node under the **Patient** input node. We refer to the first node in this lesson.

   After you complete this step, the XMap editor creates a Map statement named **Specialty to Claims/@Specialization** in the grid. This statement passes the name of the medical field of specialization to the output XML.

4. Drag the mouse from the **Name** input schema node to the **Name** output schema node.

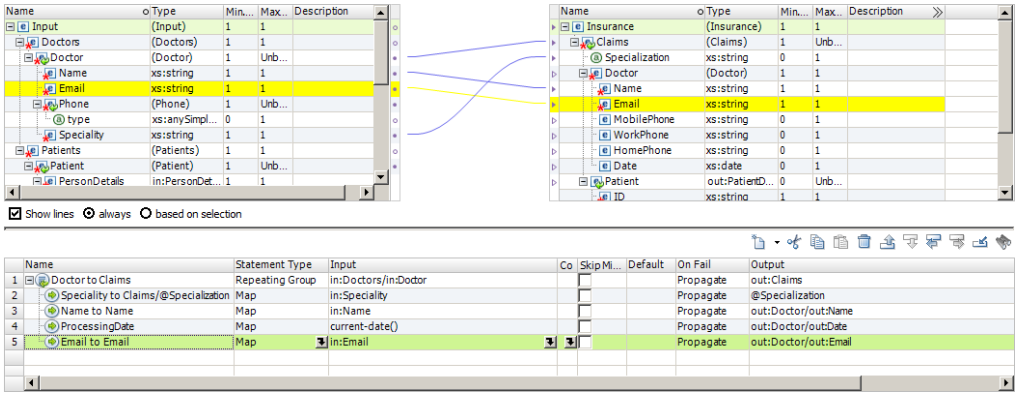
   The XMap editor creates a Map statement named **Name to Name** in the grid.

   This statement passes the name of the doctor to the output XML.

5. Drag the mouse from the **Email** input schema node to the **Email** output schema node.

   The XMap editor creates a Map statement named **Email to Email** in the grid. This statement passes the customer phone number to the output XML.

   The following image shows how the XMap links the XML input elements to the XML output in Map statements:



6. To save the transformation, click **File** > **Save**.

Step 2. Test the Transformation

Save the Data Processor transformation. Then test the transformation in the **Data Viewer** view.

1. To test the transformation, open the **Data Viewer** view.
2. Click **Run**.

The Developer tool validates and runs the transformation. If there is no error, the Developer tool shows the example file in the **Input** area. The output results appear in the **Output** panel. The following image shows details for doctors who submit claims:

Output

Name: out.xml

```
<?xml version="1.0" encoding="windows-1252"?>
<out:Insurance xmlns:out="http://www.informatica.com/B2B/XMapDemo/Hobbies/Classes">
<out:Claims Specialization="Cardiologist">
<out:Doctor>
<out:Name>Hammett Vargas</out:Name>
<out:Email>hammett.vargas@gmail.com</out:Email>
</out:Doctor>
</out:Claims>
<out:Claims Specialization="Hematologist">
<out:Doctor>
<out:Name>Janna Colon</out:Name>
<out:Email>janna.colon@gmail.com</out:Email>
</out:Doctor>
</out:Claims>
<out:Claims Specialization="Neurologist">
<out:Doctor>
<out:Name>Penelope Lowery</out:Name>
<out:Email>penelope.lowery@yahoo.com</out:Email>
</out:Doctor>
</out:Claims>
<out:Claims Specialization="Endocrinologist">
<out:Doctor>
<out:Name>Channing Cohen</out:Name>
<out:Email>channing.cohen@hotmail.com</out:Email>
```

Source | Preview

3. To debug an error, click **Show Events** to show the **Data Processor Events** view. Double-click an event in the **Data Processor Events** view in order to view further details.

Testing XMap - Tips

Use the following tips to help you validate and test an XMap more effectively.

Verify that the Data Viewer is synched with the Editor.

Before you test an XMap in the **Data Viewer** view, verify that you synch the view with the XMap editor. To verify that the **Data Viewer** view is synched, click the **Synchronize with Editor** button .

Verify the example source.

You need to chose an example source to test. To verify that you have an example source, check the **Data Viewer** view **Input** panel. The **Name** field lists the example source and the panel shows the example source content. You define the example source when create the XMap. To select or change an example source, in the XMap editor click the **Choose Example Source** icon above the input schema and browse to the example source file.

Check the output.

The Data Viewer view shows the entire output in an Output panel. You can scroll through the output file to verify that the XMap mapping logic works as you expect it to.

Check the schemas.

The XMap uses the output schema hierarchy and the XMap mapping logic to sort the output. If you see unexpected output, check the hierarchies of the input and output schemas. Check that the element characteristics are defined correctly.

Check the example source.

If the output is missing elements, check the example source file. Verify that the elements and the element characteristics are defined correctly.

Use keyboard shortcuts.

You can use keyboard shortcuts to complete tasks in the Developer tool. To view the keyboard shortcuts, click **Help** > **Key Assist**.

CHAPTER 4

Configuring XPath Expressions

This chapter includes the following topics:

- Configuring XPath Expressions Overview, 20
- Step 1. Create an XPath Function Expression, 21
- Step 2. Test the Transformation, 23
- Writing XPath Expression - Tips, 24

Configuring XPath Expressions Overview

In this lesson, you configure the **Patient_Claim** XMap with a mapping statement that passes the current date, the date you submit claims, to the output.

Lesson Concepts

Mapping statements contain fields that you can configure to customize the statement. Use the XPath editor to configure expressions. XPath is a query language used to select nodes in an XML document and perform computations.

You can configure XPath expressions in Input, Condition, and Output mapping fields. XPath expressions identify specific elements in XML documents, or check for conditions in the data. Create expressions in the XPath editor. When you click the Open button to the right of the Input, Condition, or Output field, you open the XPath editor.

The XPath editor has a Navigation panel with a function library that you can use to create XPath expressions. The functions are standard for the W3C XML Path Language. The function library also includes some functions that are specific to the Data Processor transformation.

Lesson Objectives

In this lesson, you complete the following tasks:

- Create an XPath function expression in a mapping statement that passes the current date to the output.
- Use the XPath Expression Editor to create the function expression that generates the current date.
- Validate the transformation.

Lesson Prerequisites

Before you start this lesson, verify the following prerequisites:

- You completed previous lessons in the tutorial. Use the **Patient_Claim** XMap that you created in previous lessons.

Lesson Timing

Set aside 5 to 10 minutes to complete the tasks in this lesson.

Step 1. Create an XPath Function Expression

Add a mapping statement that provides the current date as the date when claims are submitted. Use an XPath function to provide the date.

1. In the **XMap** view, add a Map statement to the **Patient_Claim** XMap, nested under the **Doctor to Claims** Repeating Group. To add a Map statement, in the XMap editor grid select the **Doctor to Claims** Repeating Group, then right-click and select **New** > **Map**.

   The Map statement is indented signifying that it is within the context of the **Doctor to Claims** Repeating Group statement. The statement iterates for each doctor with a set of details in the input.

2. To name the Map statement, double-click the **Name** field and type the name **ProcessingDate**.

3. To define the **Output** field in the **ProcessingDate** statement, drag the mouse from the **Date** output schema node to the **Output** field.

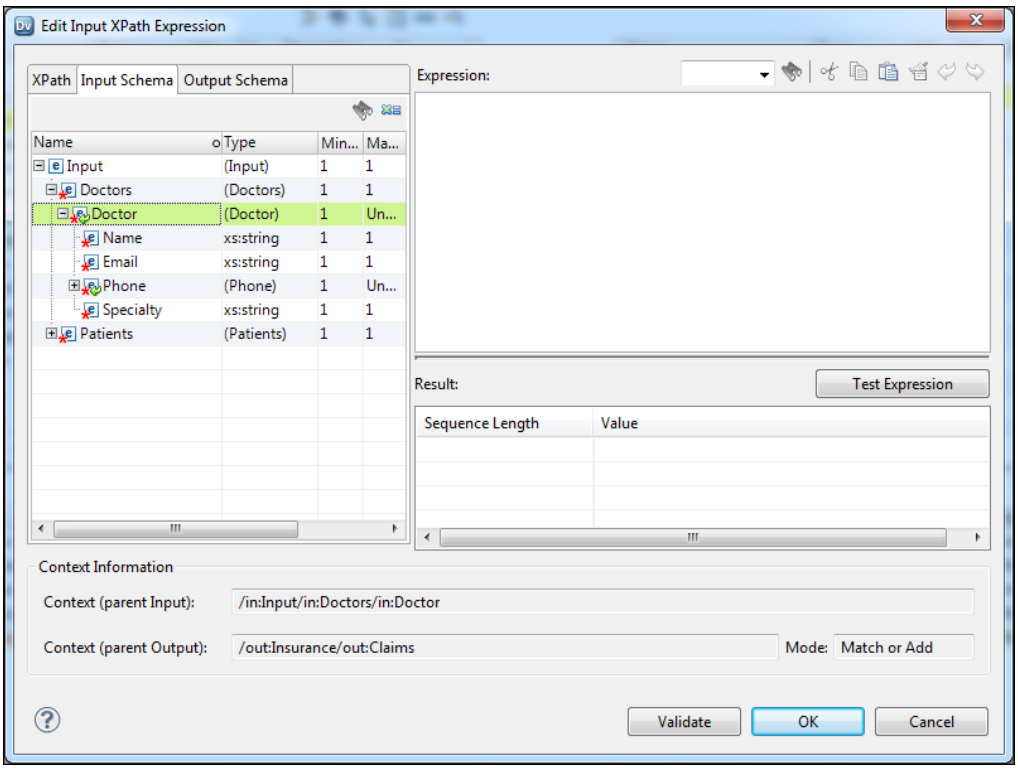
   The following image shows the mapping statement grid with the **Date** element in the **Output** field of the **ProcessingDate** statement:

	Name	Statement...	Input	Condition	Skip...	Default	On Fail	Output	Mode
1	Doctor to Claims	Repeating...	in:Doctors/in:Doctor				Propagate	out:Claims	Add
2	Speciality to Claims/@Specializ...	Map	in:Speciality				Propagate	@Specialization	Match or Add
3	Name to Name	Map	in:Name				Propagate	out:Doctor/out:Name	Match or Add
4	Email to Email	Map	in:Email				Propagate	out:Doctor/out:Email	Match or Add
5	ProcessingDate	Map					Propagate	out:Doctor/out:D...	Match or Add

   The **ProcessingDate** statement name is highlighted in red because the statement is missing an input expression.

4. To assign the current date as the input to the statement, use the XPath Expression editor to create an XPath expression in the **Input** field of the statement. To open the XPath Expression editor, click the arrow to the right of the **Input** field.

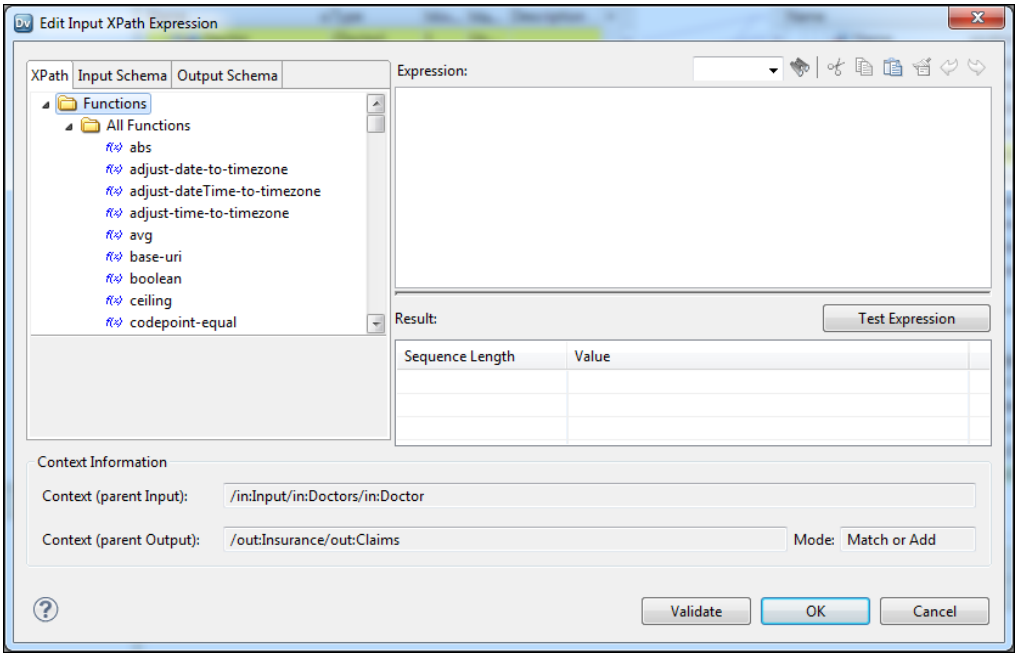
In the following image, on the left the XPath Expression editor displays the schema and function tabs, and on the right the **Expressions** pane:



Use tabs on the left to choose XPath functions, input schema nodes, or output schema nodes. Drag functions or nodes from the tabs into the **Expression** pane on the right, or type an expression directly into the pane.

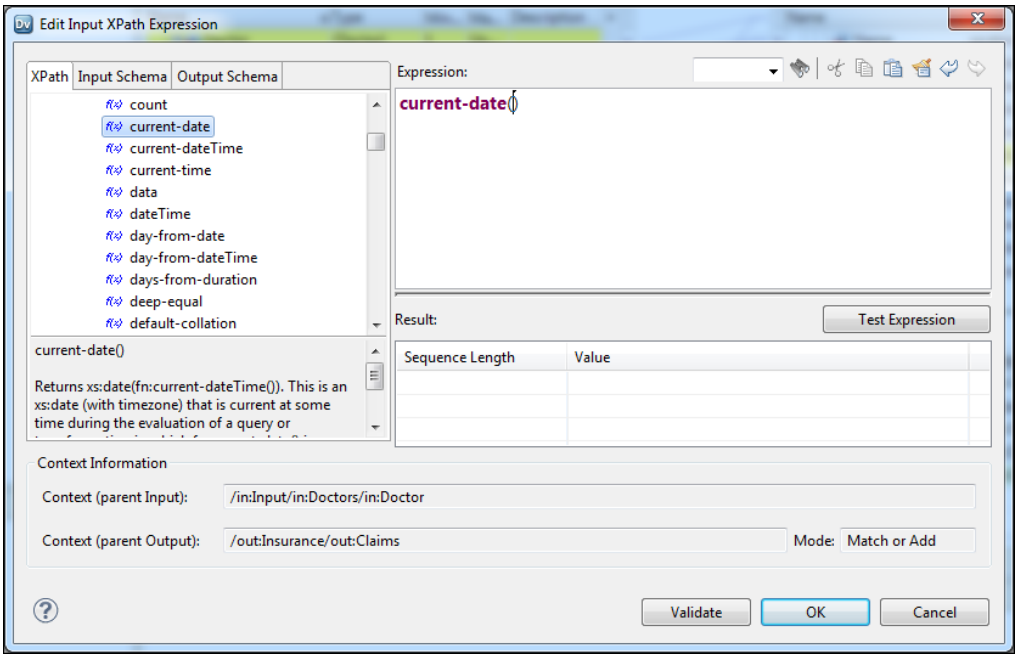
5. To select a function, open the **XPath** tab and select **Functions** > **All Functions**.

   In the following image, the XPath Expression editor displays the **Functions** tab:



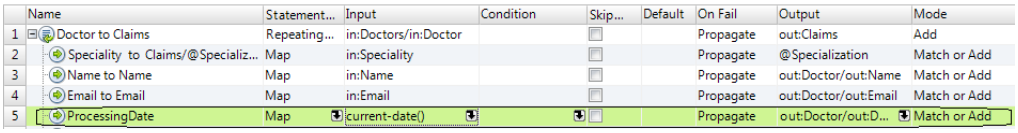
6. To select a function that generates the current date, scroll down and double-click the **current-date** function. The expression appears in the **Expression** pane.

   In the following image, the XPath Expression editor displays the **current-date** expression that you selected on the right:



7. To validate that the expression follows XPath rules, click **Validate**.
8. To enter the expression into the statement, click **OK**.

   The following image shows the XMap editor with the completed statement:



	Name	Statement...	Input	Condition	Skip...	Default	On Fail	Output	Mode
1	Doctor to Claims	Repeating...	in:Doctors/in:Doctor				Propagate	out:Claims	Add
2	Speciality to Claims/@Specializ...	Map	in:Speciality				Propagate	@Specialization	Match or Add
3	Name to Name	Map	in:Name				Propagate	out:Doctor/out:Name	Match or Add
4	Email to Email	Map	in:Email				Propagate	out:Doctor/out:Email	Match or Add
5	ProcessingDate	Map	current-date()				Propagate	out:Doctor/out:D...	Match or Add

9. To save the transformation, click **File** > **Save**.

Step 2. Test the Transformation

Save the Data Processor transformation. Then test the transformation in the **Data Viewer** view.

1. To test the transformation, open the **Data Viewer** view.

2. Click **Run**.

   The Developer tool validates the transformation. If there is no error, the Developer tool shows the example file in the **Input** area. The output results appear in the **Output** panel. The following image shows that the processing date is added to the output for each doctor:

   Output

   Name: out.xml

   ```
   <?xml version="1.0" encoding="windows-1252"?>
   <out:Insurance xmlns:out="http://www.informatica.com/B2B/XMapDem
   <out:Claims Specialization="Cardiologist">
   <out:Doctor>
   <out:Name>Hammett Vargas</out:Name>
   <out:Email>hammett.vargas@hospital.com</out:Email>
   <out:Date>2013-08-20+03:00</out:Date>
   ```

   The output is sorted according to the output schema hierarchy. The output schema categorizes the **Date** element as a sub-element for the **Doctor** element.

Writing XPath Expression - Tips

Use the following tips to help you create and test XPath expressions more effectively.

Verify that the XPath expression is valid.

To ensure that an XPath expression follows XPath rules, type the expression in the XPath Expressions editor and click **Validate**. An error message appears if the expression is not valid.

Verify that the XPath expression produces the expected results.

Ensure that the XPath expression finds or tests the nodes that you expect it to. To verify that the XPath expression tests the correct nodes, type the expression in the XPath Expressions editor and click **Test Expression**. The XPath editor displays the nodes that the expression produces in the **Result** pane. The **Sequence Length** field displays the number of matching nodes for the expression, and the **Value** field displays the XML output for the expression. If the expression contains an element that is part of a repeating group, the expression is iterated. Each line in the **Results** pane displays the results of a single iteration.

CHAPTER 5

Configuring Group Statements

This chapter includes the following topics:

- Configuring Group Statements Overview, 25
- Step 1. Create a Group Statement with Conditions, 26
- Step 2. Test the Transformation, 29

Configuring Group Statements Overview

In this lesson, you configure the **Patient_Claim** XMap to associate patients with doctors and pass patient details to the output.

Lesson Concepts

A Group statement contains a logical group of statements. A parent Group statement contains child statements. The child statements are nested underneath the Group statement in the XMap editor grid.

You can use a Group statement to provide a common context or common condition for success or failure to a group of statements. You can use a Group mapping statement if you want a set of statements to either all pass or all fail. You can also use a Group mapping statement to group a set of statements to organize and simplify an XMap grid.

A Repeating Group input is a repeating element. A Group input evaluates to a single simple element or single complex element. A complex element is an element that contains sub-elements. The Map statement input is a single simple element.

Statements contain fields that you can configure to customize the statement. The **On Fail** field determines the action taken if a statement fails. Choose one of the following options:

- Skip. If the statement fails, skip this statement. This option does not affect any related mapping logic.
- Propagate. If the statement fails, force the parent statement to also fail. If the statement is nested under a Group, Repeating Group, or Router Option statement, the entire group or option will fail and none of the statements are performed.

Lesson Objectives

In this lesson, you complete the following tasks:

- Create a Repeating Group statement to pass sets of patient details to the output. Create another Repeating Group statement that sorts the patients according to the doctor that submitted the patient claim.

- Create a Group statement to pass a single set of patient claim details for a specific patient to the output. Create another Group to pass a single set of personal details.
- Create Map statements to each pass one contact detail to the output.
- Define Map statement fields.
- Validate the transformation.

Lesson Prerequisites

Before you start this lesson, verify the following prerequisites:

- You completed previous lessons in the tutorial. Use the **Patient_Claim** XMap that you created in previous lessons.

Lesson Timing

Set aside 25 to 35 minutes to complete the tasks in this lesson.

Step 1. Create a Group Statement with Conditions

Create Repeating Group statements to process the details for patients that each doctor submits claims for. Create a Group statement to process a single set of patient details for a specific patient.

1. In the **XMap** view, add a Repeating Group statement to the **Patient_Claim** XMap. To add a Repeating Group statement, in the XMap editor grid, right-click and select **New** > **Repeating Group**.
2. To define the Repeating Group, perform the following steps:
   a. To name the statement, double-click the **Name** field and type the name **PatientDetails**.
   b. Drag the input schema node **Patient** into the Input field of the **PatientDetails** statement.

      Each time the transformation iterates a Repeating Group statement, the transformation processes a set of patient details.

   The following image shows the final **PatientDetails** Repeating Group:

	Name	Statement T...	Input	Condition	Skip Mi...	Default	On Fail	Output	Mode	Remark
1	Doctor to Claims	Repeating G...	in:Doctors/in:Doctor				Propagate	out:Claims	Add	
2	Speciality to Claims/...	Map	in:Speciality				Propagate	@Specialization	Match...	
3	Name to Name	Map	in:Name				Propagate	out:Doctor/out:Name	Match...	
4	Email to Email	Map	in:Email				Propagate	out:Doctor/out:Email	Match...	
5	ProcessingDate	Map	current-date()				Propagate	out:Doctor/out:Date	Match...	
6	PatientDetails	Repeating G...	in:Patients/in:Patient				Propagate		Add	

3. To create a claim for each appointment, compare the appointment details to the doctor specialization. Drag the **Patient** input schema node to the **Claims** output schema node.

   The editor creates a Repeating Group that provides the context for the patient data. We want to change the Repeating Group to a Group that contains a set of statement to perform for each claim in the output.
4. To define the Repeating Group, perform the following steps:
   a. To change the statement to be a Group statement, click the arrow to the right of the **Statement Type** field and select **Group**.

      When the transformation processes the Group statement, the transformation processes the set of statements nested in the Group statement.
   b. To name the statement, double-click the **Name** field and type the name **ContactDetails**.

c. Use the XPath Expression editor to edit the expression in the **Output** field for the **ContactDetails** Group statement. To open the XPath Expression editor, click the arrow to the right of the **Output** field.

d. In the **Expressions** pane, add brackets to the right of the expression to create the following expression: `out:Claims[]`

   We use brackets to create an XPath predicate that acts as a filter. We want to associate patient information with the doctor with whom the patient had an appointment. This is also the doctor who makes the insurance claim. In the next steps, we see how to do this.

e. To filter according to the doctor specialization, use the specialization element as the filter condition for the doctor appointments. To position the cursor in the expression, in the **Expressions** pane, click between the brackets in the expression `out:Claims[]`. In the **Output Schema** tab, double-click to select the **Specialization** node.

   The action creates the following expression: `out:Claims[@Specialization]`

f. To the right of the `Specialization` node in the expression, add an equals sign to create the following expression: `out:Claims[@Specialization=]`

g. To use the filter to match the output **Specialization** element to the input appointment **Speciality** element, add the **Speciality** element to the expression. Open the **Input Schema** tab and double-click the **Speciality** node that is under the **Appointment** node.

   Note: There is a **Specialty** input node under the **Doctor** input node, and a **Speciality** input node under the **Patient** input node. We refer to the second node in this lesson.

   The action creates the following expression: `out:Claims[@Specialization=../dp:input()/in:Speciality]`. The filter uses the `dp:input()` function to refer to an input node from the output statement. The expression returns nodes that match **Specialization** to **Speciality**.

h. To ensure that the claim data is matched or added for the relevant doctor, change the **Mode** to **Match or Add**.

The following image shows the final **ContactDetails** Repeating Group statement:

	Name	Statement T...	Input	Condition	Skip Mi...	Default	On Fail	Output	Mode
6	PatientDetails	Repeating G...	in:Patients/in:Patient				Propagate		Add
7	ContactDetails	Group	.				Propagate	out:Claims[@Specialization=../dp:input()/in:Speciality]	Match...

5. To transform a set of details for a specific patient, create a Group statement. Drag and drop from the **PersonDetails** input node to the **Patient** output node. In the grid, right-click the **PersonDetails to Patient** Group, and select **Demote** to nest the Group statement under the **ContactDetails** Repeating Group.

   The XMap editor creates a Group statement in the grid. This statement passes a single set of patient details to the output XML. When the Group statement is indented, this signifies that it is performed for every patient whose doctor appointment details match the speciality for that iteration. This means that the set of patient details are associated in the output with the doctor that makes the claim.

6. To find every patient for the Group statement according to the patient ID, use the XPath Expression editor to edit the expression in the **Output** field. To define the **Output** field, perform the following steps:

   a. To open the XPath Expression editor, click the arrow to the right of the **Output** field.

   b. In the Expressions pane, add brackets to the right of the expression to create the following expression: `out:Patient[]`

      We want to find input patient contact information according to the patient ID. We also assign the input patient ID data to the ID output element. In the next steps, we see how to do this.

   c. To identify the patient according to the patient ID number, filter the patient according to the **ID** node as part of the output expression. Open the **Output Schema** tab and double-click the **ID** node.

      The action creates the following expression: `out:Patient[out:ID]`

d. To the right of the expression, add an equals sign to create the following expression: `out:Patient[out:ID=]`

e. To use the filter to match the output **ID** element to the input **id** element, add the **id** element to the expression. Open the **Input Schema** tab and double-click the **id** node.

   The action creates the following expression: `out:Patient[out:ID=../dp:input()/@id]`. The statement filters patients according to ID and passes the ID to the output.

The following image shows the final Group statement:

	Name	Statement T...	Input	Condition	Skip	Mi...	Default	On Fail	Output	Mode
6	PatientDetails	Repeating G...	in:Patients/in:Patient					Propagate		Add
7	ContactDetails	Group	.					Propagate	out:Claims[@Specialization=../dp:input()/in:Speciality]	Match...
8	PersonDetails to Patient	Group	../in:PersonDetails					Skip	out:Patient[out:ID=../dp:input()/@id]	Add

7. Drag the mouse from the **Last** input schema node to the **Name** output schema node.
   The XMap editor creates a Map statement named **Last to Name** in the grid.

8. To demote the **Last to Name** statement to be within the context of the **PersonDetails to Patient** Group statement, select the statement and click **Alt-Right** as needed.

   The statement is nested within the Repeating Group and Group statements that select patients according to the filters that you defined. The patients are filtered according to the doctor that makes the claim and according to the patient ID. For each iteration of the Group statement, the statement passes the last name of the filtered patient to the output XML.

9. To combine the first and last name in the **Name** output schema node, use the XPath Expression editor to edit the expression. To get data from both the first name and last name nodes in the input, edit the **Input** field. To edit the **Input** field, perform the following steps:

   a. To open the XPath Expression editor, click the arrow to the right of the **Input** field.

   b. To select a function, open the **XPath** tab and select **Functions** > **String**.

   c. To combine the first and last names, use the `concat()` function. Scroll down and double-click `concat()`.

      The action creates the following expression: `concat()in:Name/in:Last`.

   d. To correctly position the last name, delete the close parenthesis character, then add it to the end of the expression.
      The function operates on parameters that are within the parenthesis, so we must position the name within the parenthesis. The action creates the following expression: `concat(in:Name/in:Last)`.

   e. To add the first name, open the **Input Schema** tab and double-click the **First** node.
      We want to append the first name after the last name. The action creates the following expression: `concat(in:Name/in:Lastin:Name/in:First)`

   f. To add a comma and space between the last and first name, add commas and quotation marks between the last name node and the first name node. Edit the expression to create the following expression: `concat(in:Name/in:Last,', ',in:Name/in:First)`

      We use commas to separate parameters within the concat function. The expression we created adds a space and comma between the last and first names.

   g. To test the expression, click **Test Expression**. Then, to enter the expression into the statement, click **OK**.

      When you test the expression, the **Results** panel displays all the nodes that result from the expression. With this method, you can check that the results are correct before you add the statement to the XMap. You can adjust the expression if needed.

   h. To ensure that the claim data is matched or added for the relevant doctor, change the **Mode** field to **Match or Add**.

   i. To ensure that if there is an error, the statement fails, change the **On Fail** field to **Propagate**.

The following image shows the final statement:

	Name	Statement T...	Input	Condition	Skip	Mi...	Default	On Fail	Output	Mode
6	PatientDetails	Repeating G...	in:Patients/in:Patient					Propagate		Add
7	ContactDetails	Group	.					Propagate	out:Claims[@Specialization=../dp:input()/in:Speciality]	Match...
8	PersonDetails to Patient	Group	../in:PersonDetails					Skip	out:Patient[out:ID=../dp:input()/@id]	Add
9	Last to Name	Map	concat(in:Name/in:Las ...					Propagate	out:Name	Match...

10. Drag from the **InsuranceID** input schema node to the **InsuranceID** output schema node. To demote the statement to be within the context of the **PersonDetails to Patient** Group statement, select the statement and click **Alt-Right** as needed.

    The XMap editor creates a Map statement named **InsuranceID to InsuranceID** in the grid. This statement passes a single insurance identification number to the output XML.
    The following image shows the final statement:

	Name	Statement T...	Input	Condition	Skip	Mi...	Default	On Fail	Output	Mode
6	PatientDetails	Repeating G...	in:Patients/in:Patient					Propagate		Add
7	ContactDetails	Group	.					Propagate	out:Claims[@Specialization=../dp:input()/in:Speciality]	Match...
8	PersonDetails to Patient	Group	../in:PersonDetails					Skip	out:Patient[out:ID=../dp:input()/@id]	Add
9	Last to Name	Map	concat(in:Name/in:Las ...					Propagate	out:Name	Match...
10	InsuranceID to Insuran...	Map	../in:InsuranceID		✔			Skip	out:InsuranceID	Add

11. To pass the date of the appointment to the output, drag from the **Date** input schema node to the **AppointmentDate** output schema node. To demote the statement to be within the context of the **PersonDetails to Patient** Group statement, select the statement and click **Alt-Right** as needed.

    The XMap editor creates a Map statement named **Date to AppointmentDate** in the grid.
    The following image shows the final statement:

	Name	Statement T...	Input	Condition	Skip	Mi...	Default	On Fail	Output	Mode
6	PatientDetails	Repeating G...	in:Patients/in:Patient					Propagate		Add
7	ContactDetails	Group	.					Propagate	out:Claims[@Specialization=../dp:input()/in:Speciality]	Match...
8	PersonDetails to Patient	Group	../in:PersonDetails					Skip	out:Patient[out:ID=../dp:input()/@id]	Add
9	Last to Name	Map	concat(in:Name/in:Las ...					Propagate	out:Name	Match...
10	InsuranceID to Insuran...	Map	../in:InsuranceID		✔			Skip	out:InsuranceID	Add
11	Date to AppointmentDa...	Map	../in:Appointment/in:D ...		✔			Skip	out:AppointmentDate	Add

12. To save the transformation, click **File** > **Save**.

Step 2. Test the Transformation

Save the Data Processor transformation. Then test the transformation in the **Data Viewer** view.

1. To test the transformation, open the **Data Viewer** view.

2. Click **Run**.

   The Developer tool validates the transformation. If there is no error, the Developer tool shows the example file in the **Input** area. The output results appear in the **Output** panel. The following image shows set of patients with patient details, grouped according to the doctors who submitted the claims:

Output

Name: out.xml


```
<?xml version="1.0" encoding="windows-1252"?>
<out:Insurance xmlns:out="http://www.informatica.com/B2B/XMapDemo/Hobbies/Classes">
<out:Claims Specialization="Cardiologist">
<out:Doctor>
<out:Name>Hammett Vargas</out:Name>
<out:Email>hammett.vargas@hospital.com</out:Email>
<out:MobilePhone>(702) 320-9010</out:MobilePhone>
<out:WorkPhone>(959) 731-8776</out:WorkPhone>
<out:Date>2014-04-08+03:00</out:Date>
</out:Doctor>
<out:Patient>
<out:ID>145-75-9145</out:ID>
<out:Name>Howell, Ryder</out:Name>
<out:InsuranceID>UID52298273</out:InsuranceID>
<out:AppointmentDate>2013-06-11</out:AppointmentDate>
</out:Patient>
<out:Patient>
<out:ID>838-22-4360</out:ID>
<out:Name>Barnett, Daphne</out:Name>
<out:InsuranceID>UID52298273</out:InsuranceID>
<out:AppointmentDate>2013-06-11</out:AppointmentDate>
</out:Patient>
<out:Patient>
<out:ID>573-93-7875</out:ID>
<out:Name>Wolf, Dakota</out:Name>
<out:InsuranceID>OP98II273</out:InsuranceID>
<out:AppointmentDate>2013-06-11</out:AppointmentDate>
</out:Patient>
<out:Patient>
<out:ID>247-22-5668</out:ID>
<out:Name>Ramirez, Samantha</out:Name>
<out:InsuranceID>XLT1522273</out:InsuranceID>
<out:AppointmentDate>2013-06-11</out:AppointmentDate>
</out:Patient>
```

CHAPTER 6

Configuring Router Statements

This chapter includes the following topics:

- Configuring Router Statements Overview, 31
- Step 1. Create a Router Statement, 32
- Step 2. Test the Transformation, 34

Configuring Router Statements Overview

In this lesson, you configure the **Patient_Claim** XMap to sort the telephone numbers for each doctor according to type.

Lesson Concepts

A Router statement provides alternatives for the mapping logic. The Router statement contains one or more Option statements and can contain one Default statement. When the Data Processor transformation performs a Router statement, it tests each Option nested below the Router statement. The Option statement that matches is performed.

The Option statement can contain one or more child statements of any type. If no Option statement matches, the Default statement is performed. If there is no Default statement and no Option statement matches, the Router statement fails.

If the Option statement has a condition that is true but the mapping statements inside it fail and propagate the failure, the Router fails. You can configure the mapping to skip the Router if the Router fails.

Lesson Objectives

In this lesson, you complete the following tasks:

- Create Router and Option mapping statements to provide alternatives for mapping telephone numbers of type mobile, work, and home.
- Create Map statements that pass a single telephone number to the output.
- Use the XPath editor to define input and condition fields for the Option statements. Define the condition field for each Option statement to check the type of phone an input element contains.
- Validate the transformation.

Lesson Prerequisites

Before you start this lesson, verify the following prerequisites:

- You completed the previous lessons in the tutorial. Use the **Patient_Claim** XMap that you created in previous lessons.
- You understand how to use the XPath editor.

Lesson Timing

Set aside 15 to 20 minutes to complete the tasks in this lesson.

Step 1. Create a Router Statement

Create a Router statement to check the phone numbers. Add Option statements that check if a phone number type is a mobile, work, or home phone number.

1. In the **XMap** view, add a Repeating Group statement to the **Patient_Claim** XMap. To create a Repeating Group, in the XMap editor grid select the **Doctor to Claims** Repeating Group. Right-click and select **New** > **Repeating Group**.
2. To name the Repeating Group, double-click the **Name** field and type the name **SortPhones**.
3. To evaluate the phone numbers in the input, drag the **Phone** node into the **Input** field of the Repeating Group.

   The following image shows the Repeating Group statement:

	Name	Statement...	Input	Condition	Skip...	Default	On Fail	Output	Mode
1	Doctor to Claims	Repeating...	in:Doctors/in:Doctor				Propagate	out:Claims	Add
2	Speciality to Claims/@Specializ...	Map	in:Speciality				Propagate	@Specialization	Match or Add
3	Name to Name	Map	in:Name				Propagate	out:Doctor/out:Name	Match or Add
4	Email to Email	Map	in:Email				Propagate	out:Doctor/out:Email	Match or Add
5	ProcessingDate	Map	current-date()				Propagate	out:Doctor/out:Date	Match or Add
6	SortPhones	Repeating...	in:Phone				Propagate		Add

4. To filter phone numbers by type for each doctor, create a Router. In the XMap editor grid, select the **SortPhones** Repeating Group, right-click and select **New** > **Router**.

   The following image shows the XMap editor with the Router statement:

	Name	Statement...	Input	Condition	Skip...	Default	On Fail	Output	Mode
1	Doctor to Claims	Repeating...	in:Doctors/in:Doctor				Propagate	out:Claims	Add
2	Speciality to Claims/@Specializ...	Map	in:Speciality				Propagate	@Specialization	Match or Add
3	Name to Name	Map	in:Name				Propagate	out:Doctor/out:Name	Match or Add
4	Email to Email	Map	in:Email				Propagate	out:Doctor/out:Email	Match or Add
5	ProcessingDate	Map	current-date()				Propagate	out:Doctor/out:Date	Match or Add
6	SortPhones	Repeating...	in:Phone				Propagate		Add
7	PhoneType	Router					Propagate		Match or Add

5. To name the Router statement, double-click the **Name** field and type the name **PhoneType**.
6. To filter mobile numbers, create an Option. Right-click and select **New** > **Option**.
7. To name the Option statement, double-click the **Name** field and type the name **Mobile**.
8. To make the **Mobile** statement check that the type of phone number is a mobile number, drag the mouse from the **@type** input schema node to the **Condition** field in the **Mobile** statement. To define the condition, perform the following steps:

   a. To open the XPath Expression editor, click the arrow to the right of the **Condition** field.

b. Use the XPath Expression editor to edit the expression to the following syntax: `@type="mobile"`

The following image shows the XMap editor with the Option statement:

	Name	Statement...	Input	Condition	Skip...	Default	On Fail	Output	Mode
1	Doctor to Claims	Repeating...	in:Doctors/in:Doctor				Propagate	out:Claims	Add
2	Speciality to Claims/@Specializ...	Map	in:Speciality				Propagate	@Specialization	Match or Add
3	Name to Name	Map	in:Name				Propagate	out:Doctor/out:Name	Match or Add
4	Email to Email	Map	in:Email				Propagate	out:Doctor/out:Email	Match or Add
5	ProcessingDate	Map	current-date()				Propagate	out:Doctor/out:Date	Match or Add
6	SortPhones	Repeating...	in:Phone				Propagate		Add
7	PhoneType	Router					Propagate		Match or Add
8	Mobile	Option		@type="mobile"			Propagate		Match or Add

9. Drag from the **Phone** input schema node to the **MobilePhone** output schema node.

   The XMap editor creates a Map statement named **Phone to MobilePhone** in the grid under the **Mobile** statement. The statement passes the first instance of the node to the output XML. The **Input** field contains the following expression: `.[1]`. We want to select the current node, not just the first instance. In the next step, we correct the expression in the **Input** field.

10. To correct the **Input** field, use the XPath Expression editor to edit the expression to be a single period.

    The corrected statement passes the mobile phone number in the current node to the output XML.

    The following image shows the XMap editor with the completed Option statement and Map statement:

	Name	Statement Type	Input	Condition	Skip Mi...	Default	On Fail	Output	Mode
6	SortPhones	Repeating Group	in:Phone				Propagate		Add
7	Phone Type	Router					Propagate		Match or Add
8	Mobile	Option		@type="mobile"			Propagate		Match or Add
9	Phone to MobilePhone	Map	.				Propagate	out:Doctor/out:MobilePhone	Match or Add

11. To filter work numbers, copy and paste the first Option statement to create another Option statement. Select the **Mobile** Option statement that you created, right-click and select **Copy**. Select the **PhoneType** Router statement, right-click and select **Paste**.
    A copy of the **Mobile** Option statement appears nested under the Router. The Option statement has a red highlight to indicate that you must correct the fields in order for the Option statement to work. For example, two Option statements cannot have the same condition expressions.

12. To rename the copy of the Option statement, double-click the **Name** field and type the name **Work**.

13. To make the **Work** statement check that the type of phone number is a work number, edit the **Condition** field. To define the condition, perform the following steps:

    a. To open the XPath Expression editor, click the arrow to the right of the **Condition** field.

    b. Use the XPath Expression editor to edit the expression to the following syntax: `@type="work"`

14. Rename the copied **Phone to MobilePhone** Map statement nested in the grid under the **Work** Option statement. To rename the statement, double-click the **Name** field and type the name **Phone to WorkPhone**.

15. To pass the phone number to the correct node in the output, drag the **WorkPhone** node to the **Output** field of the **Phone to WorkPhone** statement.

    This statement passes the work phone number to the output element for the work phone number. If you did not edit the statement, the transformation would pass the work phone number to the output element for the mobile phone number.

16. To filter other telephone numbers, create a Default statement. Select the **PhoneType** Router statement, right-click and select **New** > **Default**.

17. To name the Default statement, double-click the **Name** field and type the name **Other**.

18. To pass the phone number to the other phone node in the output, first copy and paste the previous Map statement to create another Map statement. Select the **Phone to WorkPhone** statement that you created, right-click and select **Copy**. Select the **Other** Default statement, right-click and select **Paste**.

    The XMap editor creates a Map statement named **Phone to WorkPhone** in the grid underneath the **Other** statement.

19. Rename the copied **Phone to WorkPhone** Map statement. To rename the statement, double-click the **Name** field and type the name **Phone to OtherPhone**.

20. To pass the phone number to the correct node in the output, drag the **OtherPhone** node to the **Output** field of the **Phone to OtherPhone** statement.

    This statement passes the other phone numbers, such as the home phone number, to the output XML.

    The following image shows the XMap editor with the completed Router, Option, Default, and Map statements:

	Name	Statement Type	Input	Condition	Skip Mi...	Default	On Fail	Output	Mode
6	SortPhones	Repeating Group	in:Phone		☐		Propagate		Add
7	Phone Type	Router			☐		Propagate		Match or Add
8	Mobile	Option		@type="mobile"	☑		Propagate		Match or Add
9	Phone to MobilePhone	Map	.		☐		Propagate	out:Doctor/out:MobilePhone	Match or Add
10	Work	Option		@type="work"	☑		Propagate		Match or Add
11	Phone to WorkPhone	Map	.		☐		Propagate	out:Doctor/out:WorkPhone	Match or Add
12	Other	Default			☐		Propagate		Match or Add
13	OtherPhone	Map	.		☐		Propagate	out:Doctor/out:OtherPhone	Match or Add

21. To save the transformation, in the Developer tool click **File** > **Save**.

Step 2. Test the Transformation

Save the Data Processor transformation. Then test the transformation in the **Data Viewer** view.

1. To test the transformation, open the **Data Viewer** view.
2. Click **Run**.

   The Developer tool validates the transformation. If there is no error, the Developer tool shows the example file in the **Input** area. The output results appear in the **Output** panel. The following figure shows that the contact details per doctor include phone numbers that are sorted by type:

Output

Name: out.xml

```
<?xml version="1.0" encoding="windows-1252"?>
<out:Insurance xmlns:out="http://www.informatica.com/B2B/XMapDen
<out:Claims Specialization="Cardiologist">
<out:Doctor>
<out:Name>Hammett Vargas</out:Name>
<out:Email>hammett.vargas@hospital.com</out:Email>
<out:MobilePhone>(702) 320-9010</out:MobilePhone>
<out:WorkPhone>(959) 731-8776</out:WorkPhone>
<out:Date>2013-08-20+03:00</out:Date>
</out:Doctor>
<out:Patient>
<out:ID>145-75-9145</out:ID>
<out:Name>Howell</out:Name>
<out:Company>Mutual Insurance</out:Company>
<out:InsuranceID>UID52298273</out:InsuranceID>
<out:AppointmentDate>2013-06-11</out:AppointmentDate>
</out:Patient>
```

APPENDIX A

Frequently Asked Questions

This appendix includes the following topic:

- Frequently Asked Questions, 35

Frequently Asked Questions

The following frequently asked questions can help you configure and use an XMap more effectively.

If you drag between schema nodes to create a mapping statement, do you need to define the fields?

The XMap editor defines the fields based on the hierarchical context of the linked nodes. If you want to change the context or add mapping statement logic, you can edit the fields.

How can you use the Events View to debug the XMap?

To find the mapping statement in the XMap editor grid that correspond to events in the Events View, use the event information. An event lists the related index and mapping statement name. Use the mapping statement name to find the mapping statement in the XMap editor grid. To find the node that relates to the mapping statement, right-click the statement and select **Show In Schema Trees**. The editor highlights the relevant nodes.

Can you run an XMap from other Data Processor transformation components?

To call an XMap from a Parser, Serializer, or Mapper use the Run XMap action. You can also use the Run XMap action to call one XMap from another XMap.

Can you import a Data Processor transformation with an XMap into a PowerCenter® Unstructured Data Transformation?

Yes. PowerCenter requires a B2B Data Transformation XMap license.

APPENDIX B

Glossary

Data Processor event

An occurrence during the execution of a Data Processor transformation.

document processor

A component that operates on a document as a whole, typically performing preliminary conversions before parsing.

editor

A window where you can create and modify a transformation.

event log

A report of events that occurred during the execution of a transformation.

Events view

A window that displays the events that occur during a transformation.

example source document

A sample of the documents that a Data Processor transformation processes.

Explorer view

A window that displays the projects and files in the workspace.

Model Repository Service

An application service in the Informatica domain that runs and manages the Model repository. The Model repository stores metadata created by Informatica products in a relational database to enable collaboration among the products.

output document

A document that is the result of a Data Processor transformation.

perspective

A predefined layout of the Eclipse Workbench.

predicate expression

An expression that filters the data in a mapping. A predicate expression returns true or false.

preprocessor

A document processor used to perform an overall modification of a source document, before the main transformation.

project

A framework within which you can define transformations. A project is stored in a specified folder, although certain project files can also exist outside the folder.

project properties

Options for the behavior of a project. The options control features such as the input and output encodings, the authentication support, and the XML validation.

schema

A definition of the elements, attributes, and structure used in XML documents. The schema conforms to the World Wide Web Consortium XML Schema standard, and it is stored as an `*.xsd` file.

source document

A document that is the input of a Data Processor transformation.

startup component

The runnable component that Data Transformation starts first when it runs a Data Processor Transformation.

transformation

A repository object in a mapping that generates, modifies, or passes data. Each transformation performs a different function.

view

A window within Data Transformation Studio, which displays data about a project and lets a user perform limited operations on the data.

XMap

A Data Processor transformation object that maps an XML input document to another XML document.

XPath

A query language used to select nodes in an XML document and perform computations.

XSD schema file

An `*.xsd` file containing an XML schema that defines the elements, attributes, and structure of XML documents.