

TIBCO ActiveMatrix® BPM Mediation Component Development

Software Release 4.2

August 2017

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 2010-2017 TIBCO Software Inc. All rights reserved.

TIBCO Software Inc. Confidential Information

Contents

Figures	8
TIBCO Documentation and Support Services	9
Introduction to Mediation	11
Mediation Flows	12
Message Exchange Patterns	13
Mediation Flow Interfaces	13
Planning Target and Mediation Interfaces	14
Paths in a Mediation Flow	15
Mediation Tasks	16
Mediation Exchange	17
Designing Mediation Flows	18
Working with Mediation Flows	20
Starting the Mediation Flow Wizard	20
Creating a New, Empty Mediation Flow	20
Creating New Mediation Flows from Existing Web Services	21
Editing Mediation Flow Editor Preferences	24
Working with Mediation Flow Properties	25
Validation of Message	25
Adding a Mediation Flow Property	25
Deleting a Mediation Flow Property	26
Working with Interfaces	26
Adding Interfaces to Mediation Flows	26
Deleting Interfaces from Mediation Flows	27
Moving Mediation Interfaces	27
Using the AutoMediate Feature	27
Creating Local WSDL Files	28
Supported Policies	28
Working with Mediation Paths	28
Changing Mediation Paths	29
Deleting Mediation Paths	29
Working with Message Context Properties	29
Context Parameters	30
Mediation Context Parameters	30
Undeclared Fault Headers	32
Context Parameters in Mediation Components	32
Adding Context Parameters	32

Deleting Context Parameters	33
Defining the Scope of Context Parameters	33
Working with Exchange Variables	34
Defining Exchange Variables	34
Setting Exchange Variable	35
Creating Simple Schemas	36
Working with Tasks	37
Adding a Task to a Path	37
Deleting a Task From a Path	38
Mediation Components	38
Invoking an Operation	39
Configuring Invoke Operation Tasks	39
Logging Mediation Exchange Information	41
Mediation Appenders and Loggers	41
Configuring a Log Task	42
Routing Messages in a Mediation Flow	44
Paths and Route Tasks	45
Defining a Route	46
Adding Routing Cases	47
Specifying Case Targets in the Decision Table	47
Modifying Case Names	48
Modifying Destinations	48
Moving Cases in the List	48
Deleting Cases	48
Nesting Multiple Route Tasks	49
Adding and Deleting Variables	50
Mapping Data to Variables	51
Routing Conditions	51
Editing Route Task Conditions	52
Conditions for XPath Route Tasks	53
Changing Route Tasks to XPath Route Tasks	53
Transforming Tasks	55
Example of Transformation	56
Basic Mapping	56
Using XPath Editor	57
Data Contribution to the Mediation Exchange	58
External Stylesheets for Data Transformation	58
Specifying an External Stylesheet for Data Transformation	59
Schema Components	60

Context Panel	61
Message Panel	62
Data and Function Tabs	63
TIBCO XPath Functions	64
Creating Custom XPath Functions	65
Exporting Custom XPath Functions	66
Deploying Custom XPath Functions	67
Testing Custom XPath Functions	67
Mapper Toolbar Buttons	67
Right-Click Menu in the Message Panel	68
Surrounding a Component With a Choose Statement	69
If Statements	70
For Each Statements	71
Adding a Variable to a Mapping	71
Managing Mappings	72
Repairing Incorrect Mappings	73
Mapping an Empty Complex Type	73
Using XPath	73
Transforming XML with Related Tasks	76
Querying a Database	77
JDBC Resource Templates	77
Defining a Resource Template	77
Configuring a JDBC Driver	78
Registering a JDBC Driver	78
Configuration Tabs of the Query Database Task	79
Dynamic Requests	82
Service Providers for Dynamic Composite References	82
Configuring Dynamic Binding	83
Configuring Dynamic Target Interfaces	84
Pattern Variables Usage	84
Dynamic Reference Task Setting	85
General Tab Configuration	87
Input Specification	88
Configuring Dynamic References in Composite	89
Creating and Deploying Composites Used By Dynamic Binding	89
Replying to Messages	90
Fault Processing in a Mediation Flow	92
Throwing Faults in Mediation Flows	93
Fault Paths	93

Catch Fault Configuration	94
Catching Faults from the Mediation Flow	95
Sending Faults to the Invoker	96
Custom Mediation Tasks	97
Migrating Custom Mediation Tasks	97
Eclipse Plug-in Reference	98
Support Files	99
Creating the Model Plug-in	99
Creating the UI Plug-in	102
Creating the Runtime Plug-in	103
Writing Custom Mediation Code	104
Accessing Task Input/Output Schema	105
Modifying the Mediation Task Data	105
Defining Model Attributes	106
Custom Mediation Task Categories	106
Thrown Faults	107
Runtime Exceptions	107
Installing Custom Mediation Tasks	107
Deploying Custom Mediation Tasks	107
Testing Custom Mediation Tasks	108
Reference	109
Catch Fault	109
End Mediation	109
Generate Reply	111
Handle Reply	112
Invoke Operation	112
Log	113
Information for Standard Log Messages	114
Information for Custom Log Messages	115
Parse XML	116
Query Database	118
Render XML	121
Route Task	122
Send Fault	125
Set Context	125
Set Dynamic Reference	126
Set Exchange Variable	129
Throw Fault	130
Transform	130

Validate XML	132
XPath Route	134
TIBCO AutoMediate Command-Line Tool	137
AutoMediate Command-Line Tool Flow	137
Running the AutoMediate Command-Line Tool	138
AutoMediate Command Syntax and Options	139
AutoMediate ANT Command Syntax and Options	142
Introduction to gXML Applications	143
Developing gXML Applications	143
Implementing GxApplication	143
Implementing GxCatalog	145
Implementing GxResolver	145
Injecting DOM	147
gXML Recipes	148
Parsing a Character Stream and a Byte Stream	148
Constructing a Data Model Tree Programmatically	149
Validating	156
Navigation	157
Mutation	159
Serialization	160
XPath	161
XSLT	164
XQuery	168
Validation	172

Figures

Mediation Components	11
Mediation Example	12
The Mediation Flow Editor	12
Paths in a mediation flow for each message exchange pattern	15
Mediation exchange information	17
An example of using Route Task	44
Output path for a route	45
Fault path for a route	46
Adding a routing case	47
Mapping values to variables	51
Routing conditions for Route task	52
Routing with more than one variable	53
The Input tab of a Transform task	55
A travel reservation mediation flow: the input path	56
A basic mapping example	57
The XPath Editor Window	57
Dragging a data element into a function	58
An example of choose statement	70
Example of If statement	70
Example of For Each statement	71
An example of adding a variable to a mapping	72
Static and dynamic binding	82
Service providers and pass-through composites	83
Dynamic and static target interfaces	84
An example of pattern variables	85
Sending a reply message	90
An example of fault path	92
Configuring a catch fault task	94
Removing specific faults from the target operation fault path	95
Catch fault task for the mediation flow	95

TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, visit:

<https://docs.tibco.com>

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site.

The following documents form the documentation set:

- *Concepts*: Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of the platform. The other manuals in the documentation set assume you are familiar with the information in this manual.
- *Development Tutorials*: Read this manual for a step-by-step introduction to the process of creating, packaging, and running composites in TIBCO Business Studio.
- *Composite Development*: Read this manual to learn how to develop and package composites.
- *Java Component Development*: Read this manual to learn how to configure and implement Java components.
- *Mediation Component Development* : Read this manual to learn how to configure and implement Mediation components.
- *Mediation API Reference* : Read this manual to learn how to develop custom Mediation tasks.
- *Spring Component Development* : Read this manual to learn how to configure and implement Spring components.
- *WebApp Component Development* : Read this manual to learn how to configure and implement Web Application components.
- *Administration Tutorial*: Read this manual for a step-by-step introduction to the process of creating and starting the runtime version of the product, starting TIBCO ActiveMatrix servers, and deploying applications to the runtime.
- *Administration*: Read this manual to learn how to manage the runtime and deploy and manage applications.
- *Hawk ActiveMatrix Plug-in User's Guide*: Read this manual to learn about the Hawk plug-in and its optional configurations.
- *Installation and Configuration*: Read this manual to learn how to install and configure the software.
- *Release Notes*: Read this manual for a list of new and changed features, steps for migrating from a previous release, and lists of known issues and closed issues for the release.

The documentation for the following features is installed separately:

- TIBCO ActiveMatrix Implementation Type for C++
- TIBCO ActiveMatrix Binding Type for EJB
- TIBCO ActiveMatrix Binding Type for Adapters
- TIBCO ActiveMatrix Implementation Type for TIBCO Adapters
- TIBCO ActiveMatrix Implementation Type for Microsoft CLR
- TIBCO ActiveMatrix Binding Type for REST

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

<https://community.tibco.com>

Introduction to Mediation

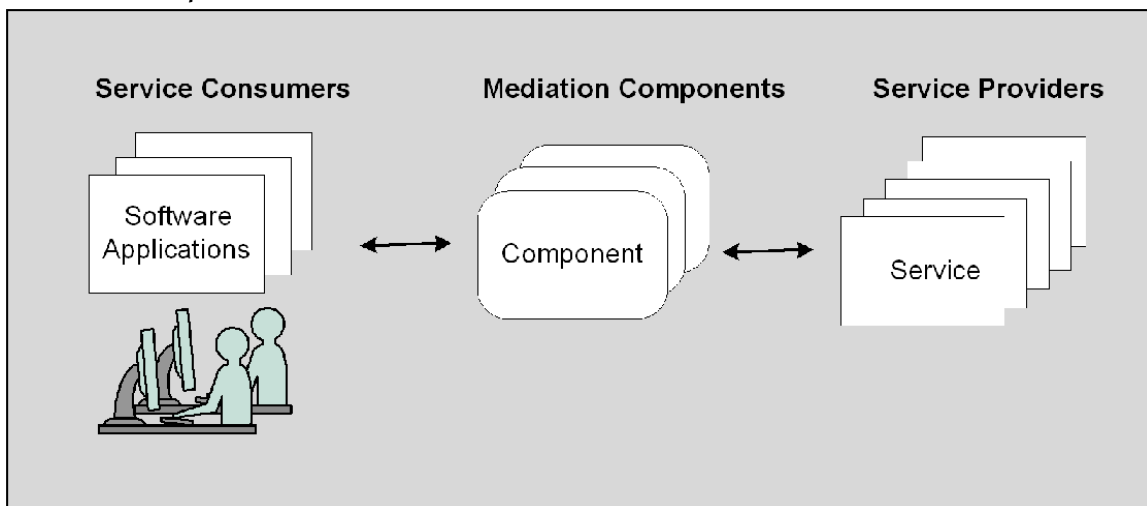
Mediation involves virtualizing and managing service interactions between communicating participants.

Mediation is part of a Service Oriented Architecture (SOA) for applications. ActiveMatrix implements a component-based platform to implement SOA within an enterprise.

Advantages of a mediation component are:

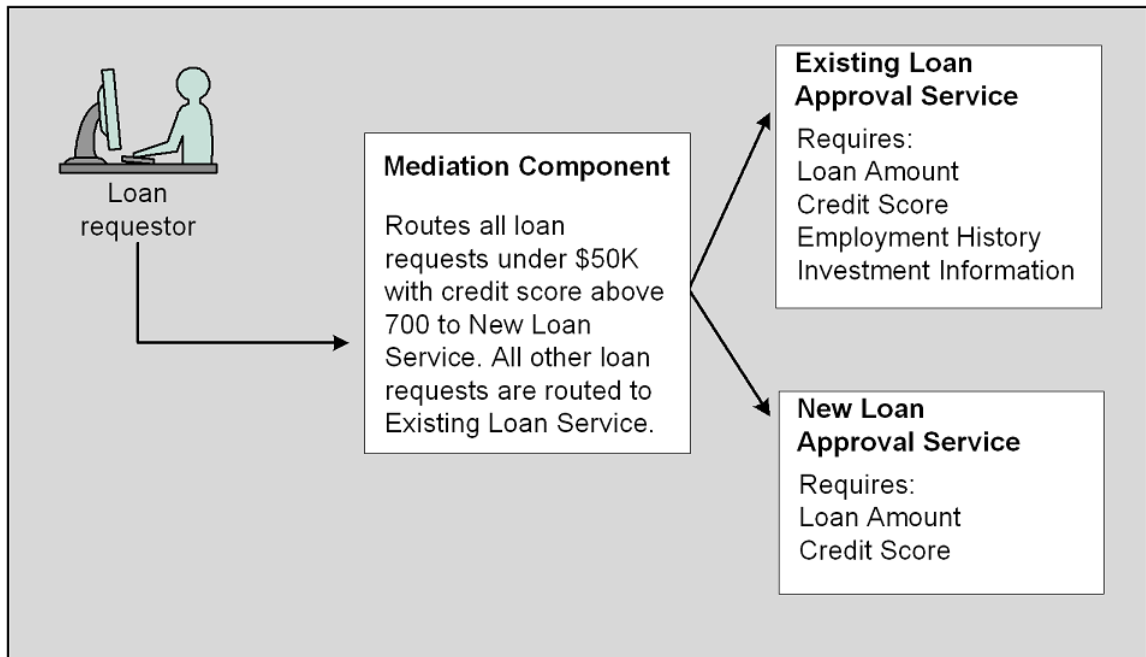
- Shields service consumers from the service provider's physical location from both the design-time and run-time perspective.
- Is responsible for delivering requests to a service provider.
- Provides a mechanism for handling changing service requirements.
- Service providers can respond to requests delivered by mediation components without needing to know the point of origin of the message.

Mediation Components



The below example shows a mediation component that provides approvals for loan applications. The existing loan service might require credit scores, loan amount, employment history, and so on. To enable quick turnaround for smaller loans, you might want to provide a new service that approves loan requests for under \$50,000 for all applicants with credit scores above 700.

Mediation Example



Instead of rewriting your existing service to handle new types of requests, a mediation component can accept requests that contain information from the loan requestor and then submit the request to the appropriate service for approval.

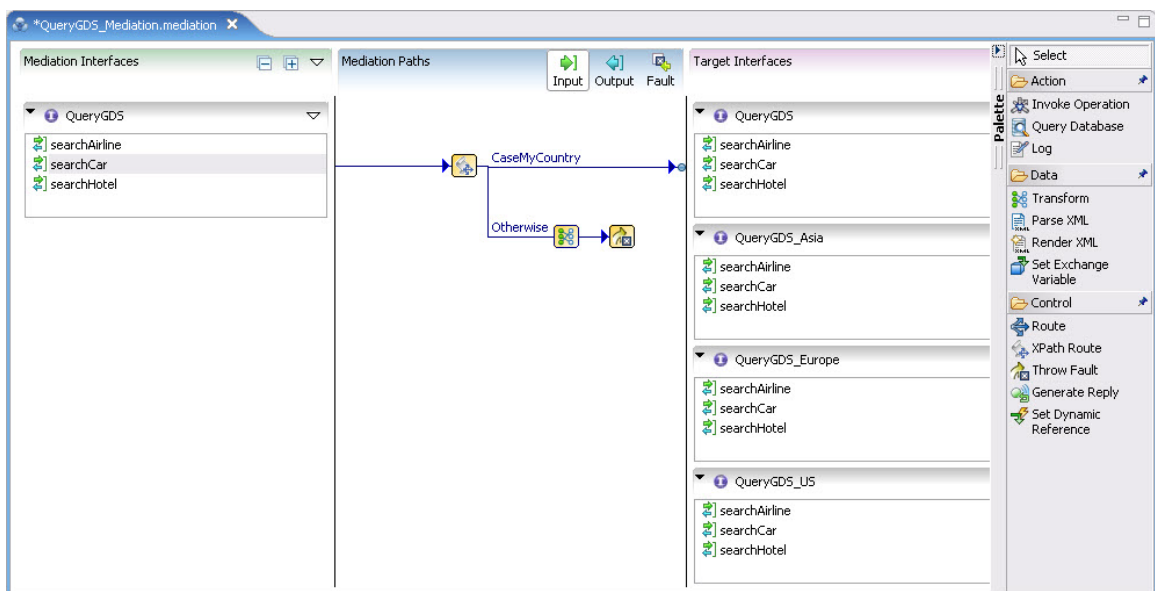
Using the Mediation Flow Editor, you can easily create mediation components that operate within the SOA-based ActiveMatrix platform.

Mediation Flows

A mediation flow is a graphical representation of the business logic for a mediation component.

Mediation flows are created and managed within the Mediation Flow Editor, in TIBCO Business Studio. Mediation flow resources are stored in the Mediation Flows folder within a ActiveMatrix SOA project. See the Composite Development Guide for more information about creating and managing ActiveMatrix SOA Projects.

The Mediation Flow Editor



A mediation flow includes several parts:

- **Mediation interfaces:** One or more mediation interfaces provide the interface for the mediation components that you expose to consumers of your applications.
- **Mediation paths:** Depending on the message exchange pattern of the mediation interface, there can be an input, output, and fault path from each operation in the mediation interface to operations in the target interfaces.
- **Target interfaces:** These interfaces to existing services in your enterprise provide implementation of the operations for the associated mediation operations.
- **Mediation tasks:** You can place mediation tasks, such as Log or Route, on mediation paths to perform business logic your application requires.

Message Exchange Patterns

A message exchange pattern (MEP) is a template that describes the message exchange pattern between two communicating parties.

Mediation flows support two web service MEP for mediation and target operations:

- **One-way (in-only):** A message consumer sends a message to a provider.
In this exchange, the mediation flow allows only an input path from the mediation operation. No output path is used. Fault paths exist to handle any errors produced by mediation tasks executing on the input path. To terminate the mediation of a one-way operation without invoking a target operation, use the [End Mediation](#) task.
- **Request-response (in-out):** A message consumer sends a message to a provider, and the provider sends a response message back to the consumer.

In this exchange, the mediation flow has three paths:

1. An input path for the message from the consumer to the provider
2. An output path for the reply message
3. A fault path for any faults that are encountered during processing



To mediate different operations with a target operation of a different message exchange pattern, use the Invoke Operation and Generate Reply mediation tasks. See [Generate Reply](#) and [Invoke Operation](#) for more information.

Mediation Flow Interfaces

Web Service Description Language (WSDL) files define the interface to a web service.

WSDL is a standard maintained by the World Wide Web Consortium; it is beyond the scope of this guide to describe WSDL syntax and functionality in detail. You can learn more about WSDL from <http://www.w3.org/TR/wsdl>; commercial publications about Web Services and WSDL files are also available.

Mediation flows have two types of interfaces:

- **Target interfaces** are interfaces to the actual services that make up your enterprise application. These interfaces appear on the right side of a mediation flow in the Mediation Flow Editor.
- **Mediation interfaces** are interfaces that you expose to the consumers of your services. Mediation interfaces can have the same number and type of operations as target operations, or they can be different from the target interfaces. Mediation interfaces appear on the left side of a mediation flow in the Mediation Flow Editor.

See [Mediation Flows](#).

Interfaces (also called port types) in mediation flows are references to abstract web services that a WSDL file defines—interfaces in a mediation flow do not have concrete bindings. The WSDL files you use in a mediation flow might have concrete bindings, but the mediation flow is concerned only with receiving the message from the mediation operation, processing the message, and forwarding it to its target operation. Binding occurs when a mediation flow is placed into an ActiveMatrix mediation component, using the Composite Editor. See [Mediation Components](#) for more information about components.

You can add the same interface more than once to either the mediation or target interface list. Adding the same interface several times to the mediation interfaces list enables you to offer the same interface to consumers with different mediation implementations.

This functionality can be used to offer different qualities of service to different consumers of the service. Adding the same interface several times to the target interfaces list enables you to bind the same interface to different providers, if you have more than one provider of the same service.

Planning Target and Mediation Interfaces

Designing a mediation flow requires planning how services will be exposed by mediation interfaces.

The requirements of the application will drive the design of the mediation flows. There may be a one-to-one mapping of target and mediation interfaces, or you may expose mediation interfaces that are very different from your target interfaces.

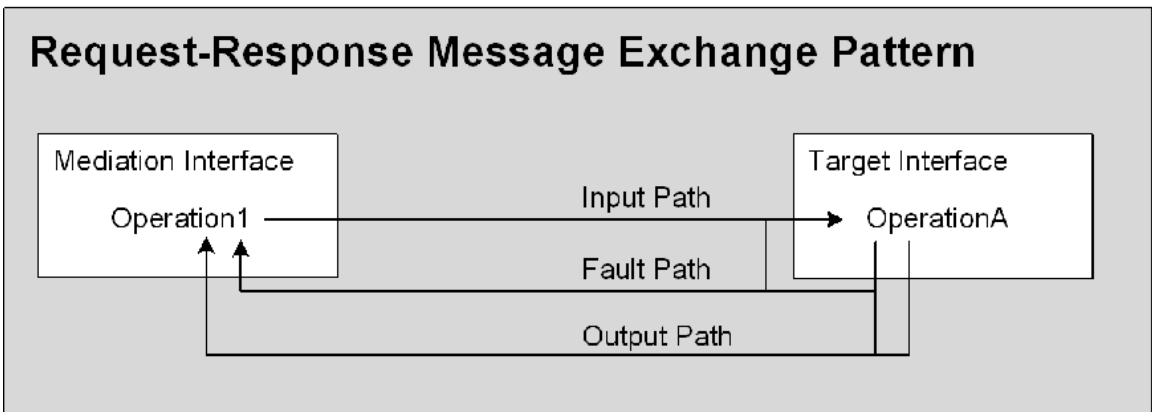
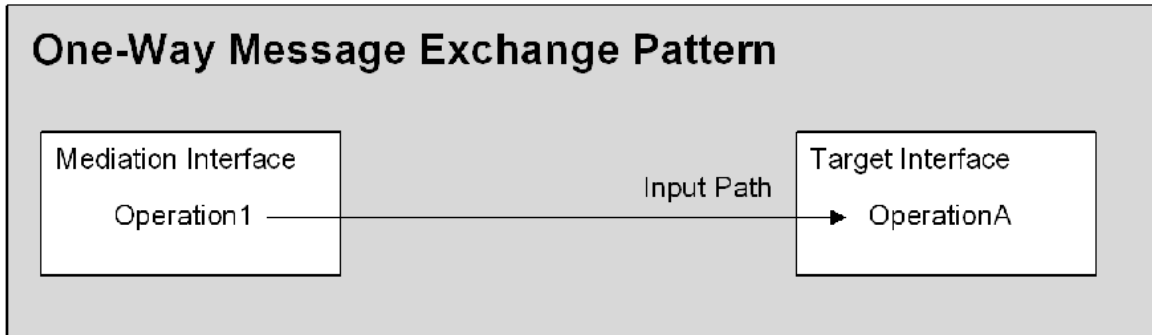
For example, you may have target services that are offered from a third party and therefore cannot change the target interfaces. In this case, if you want to offer a service that uses the third-party services but has different operations and message schemas, you must create your own interface/WSDL file describing the service to offer, and use that interface file as the mediation interface.

Before starting your project, consider the requirements of your application, plan and develop the required interfaces, and determine how the mediation interfaces will use the target interfaces.

Paths in a Mediation Flow

Each incoming message for an operation in a mediation interface follows an input path to a target operation, or a task that terminates the input path. Depending on the message exchange pattern, there could also be an output path for reply messages and a fault path for fault messages.

Paths in a mediation flow for each message exchange pattern



For operations that use the one-way message exchange pattern, there is only an input path from the mediation operation to the target operation. Operations that use the request-response message exchange pattern have an input path, an output path, and a fault path. Fault paths handle faults wherever they occur in a mediation flow—either during processing within the mediation flow, or during processing by the target operation.

The Mediation Flow Editor enables you to view the input, output, and fault paths for an operation by selecting the mediation operation, and then clicking on the appropriate button in the mediation paths area of the editor. Only the path for the currently selected mediation operation appears in the mediation paths area.

When the input path for a mediation operation is defined or changed, the output and fault paths are automatically changed to reflect the input path. Output or fault messages must be returned to the original invoker, so that the input, output, and fault paths are automatically kept compatible.

You can use Route tasks to divide a mediation path into multiple sub-paths to potential target operations. Route tasks allow the mediation path to be split into multiple sub-paths to potential target operations. While the path shows multiple potential destinations, each message is only sent to one destination. The path in the Mediation Flow Editor is like a map that describes the potential places where a message can go. When the mediation flow is executed, however, each message travels to only one target operation.

You can use multiple, nested route tasks to send a single message to a target in several different ways. Rather than using a single route task with compound conditions, the use of nested routes enables you to make complex routing decisions that are easier to follow.

Mediation Tasks

You can place mediation tasks on input, output, or fault paths, to perform business logic required by your application.

For example, if the schema of the input message of your mediation operation does not match the schema of the input message of the target operation, you can use a Transform task to change the schema to the desired format.

The Mediation Flow Editor includes a variety of mediation tasks:

- **Invoke Operation:** Enables you to invoke an operation of an interface in the target interface list during processing of an input, output, or fault path. For example, you can invoke an operation on the input mediation path and use the data in the reply message in subsequent tasks in the input path before the mediation flow invokes the specified target operation. See [Invoking an Operation](#).
- **Query Database:** Performs a SQL SELECT statement on a database. The task can specify one or more tables in the FROM clause of the SELECT statement, one or more columns to return in the SELECT list, and one or more conditions in the WHERE clause. Optionally, you can specify the maximum number of rows to return. See [Querying a Database](#).
- **Log:** Writes information to the log file. You can use this task for auditing, security, or other purposes. See [Logging Mediation Exchange Information](#).
- **Transform:** Takes information from the mediation exchange (described in Mediation Exchange on page 10) and changes it to the appropriate format. See [Transform Tasks](#).
- **Parse XML:** Used when you have an XML document stored in a string or binary field. This task produces a tree representation of the XML that can be used by subsequent tasks in the mediation flow. This task can be paired with the Render XML task to convert the parsed XML back into a string or binary field for transmission within a message. See [Parse XML](#).
- **Render XML:** Converts an XML tree for a specified schema to a string or binary element that contains the XML document. This task can be paired with the Parse XML task to convert the parsed XML back into a string or binary field for transmission within a message. See [Render XML](#).
- **Validate XML:** Provides validation of XML messages using an XML Schema that is configured at design time or specified dynamically at runtime. Validation errors may be caught and handled in the current flow path, or handled by the fault catch mechanism. See [Validate XML](#).
- **Set Context:** Provides a way to set HTTP header values or JMS user property values of the operations within a mediation flow. See [Working with Message Context Properties](#) and [Set Context](#).
- **Set Exchange Variable:** Sets the value of the items within the exchange variable. The Input tab of the Set Exchange Variable task is a mapper panel that enables you to set the exchange variable for the currently selected operation. See [Setting the Exchange Variable](#), and [Set Exchange Variable](#).
- **Route and XPath Route:** Route tasks enable you to specify more than one potential destination for messages sent by a mediation operation. The message is sent to the appropriate target operation based on criteria you specify. In Route tasks, the criteria for routing conditions are simple comparison operations. XPath Route tasks are similar to Route tasks, but you can specify more complex criteria for routing conditions. See [Routing Messages in a Mediation Flow](#).
- **Throw Fault:** Stops processing in the current mediation flow and transfers control to the fault path. This task is useful if a mediation operation is deprecated and you want to return a fault to the requestors of the operation. This task is also useful if you want to specify fault conditions for Route or XPath route tasks. See [Fault Processing in a Mediation Flow](#) for more information about faults and the Throw Fault task.

- **Generate Reply and Handle Reply:** In some situations, you might want to send a reply message to a consumer without invoking the target operation. The Generate Reply and Handle Reply tasks enable you to bypass the target operation and send reply messages to the consumer directly from the mediation flow. See [Replying to Messages](#).
- **End Mediation:** Ends a one-way (in-only) message exchange pattern for operations that don't involve a response. End Mediation includes a message re-delivery feature, so that you can request that a message be re-delivered if it encounters a fault during processing. See [End Mediation](#).
- **Set Dynamic Reference:** Provides the values needed for resolving a service provider in a dynamic target interface. Each Set Dynamic Reference task sets the value of the service provider for the specified dynamic target interface. The value is then used by the next service invocation that refers to that dynamic target interface. See [Set Dynamic Reference Task](#).

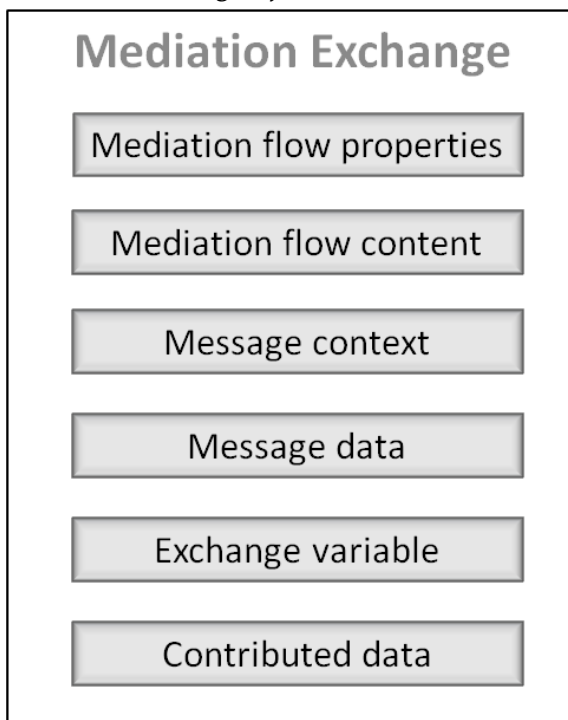


You can extend the functionality of mediation flows by creating your own Custom tasks. See [Custom Mediation Tasks](#).

Mediation Exchange

When a mediation operation receives a message, a mediation exchange is created to hold information related to the message and the mediation flow. Information in the mediation exchange is available to tasks in the mediation flow.

Mediation exchange information



The mediation exchange consists of this information:

- **Mediation flow properties:** You can define properties on a mediation flow to store information used within the flow. For example, you might create a property to store currency exchange rates, or calendar holidays for system down time.
- **Mediation flow context:** Includes information such as component name and mediation flow information, if the Mediation Flow Context option is set on the Advanced tab of the mediation operation's Properties view. See [Working with Message Context Properties](#).

- **Message context:** The context of the message sent to the mediation operation. Message context includes information about the message transport (for example, HTTP or JMS message headers) and security context information about the message. You can use the Set Context task to set HTTP header values and JMS user property values within a mediation flow — see [Working with Message Context Properties](#) and [Set Context](#).
- **Message data:** Content of the message. The content of this item depends on the processing within a mediation flow. For example, for input paths this component contains the schema of the input message of the mediation or the target operation. For output paths, this component contains the schema of the reply message of the mediation or the target operation. Similarly, for fault paths this component contains the schema of the fault message.

Some mediation tasks, such as Transform, can change the contents of the message data.

- **Exchange variable:** A defined schema to hold data that persists through all paths of a mediation operation (input, output, and fault paths). You can use any schema stored in the project to define the structure of the exchange variable. The value of the variable is set during execution of the mediation path with the Set Exchange Variable task. See [Working with Exchange Variables](#) and [Set Exchange Variable](#).
- **Contributed data:** Mediation tasks, such as the Transform task or a custom mediation task, can add — contribute — data to the mediation exchange. When the data is added, subsequent tasks can access each task's added data. An option on some mediation tasks enables you to specify whether you want the task to change the existing message data in the mediation exchange, or place the results of the task into a new data item in the mediation exchange.

Designing Mediation Flows

You can design mediation flows from the top down, or from the bottom up. That is, you can start with interfaces and mediation flows, or you can start by designing composites and components.

If you start with interfaces, you can create mediation flows from the interfaces. If you start with components, you can assign a mediation flow as the implementation of the component after specifying the services and references in the Composite editor.

1. Create an ActiveMatrix SOA project and import the WSDL files.
2. Virtualize interfaces.
 - a. Create a mediation flow.
 - b. Specify the mediation and target interfaces.
 - c. Create mediation paths.

3. Select a mediation patterns.

The most cited ESB (mediation) patterns are these:

- VETO (Validate, Enhance, Transform, Operate)
- VETRO (Validate, Enrich, Transform, Route, Operate)

Mediation provides the Validate, Transform and Route tasks.

The Enrich task can be achieved using the Query DB task, the Invoke Operation task, or a customer-created task.

The Operate task makes the target service call.

4. Configure mediation patterns using tasks.
 - Add and remove tasks.

- Configure task properties.
 - Handle faults.
5. Bind and deploy.
- Create composite and components.
 - Specify bindings.
 - Assemble and run.



Before you package and deploy your project, ensure that all validation errors are resolved. An error icon appears on the operation name of mediation interfaces with errors.

Errors occur because of an invalid configuration. Each error is logged on the Problems tab of the mediation flow.

For more information about the process of designing a mediation flow, consult these resources:

- *Composite Development Guide* describes the first step in the process, creating the project and obtain the interfaces.
- [Working with Mediation Flows](#) describes steps 2 through step 4a in more detail.
- *Composite Development Guide* describes how to create service assemblies for deployment and execution.
- *Administration Guide* describes how to deploy and run your project.

Working with Mediation Flows

You use the mediation flow wizard to create new mediation flows and mediation flows from existing web services.

Before creating mediation flows, you should have at least one WSDL file that defines the target interface that you plan to mediate. For more information about folders in ActiveMatrix SOA projects, see the Installation.

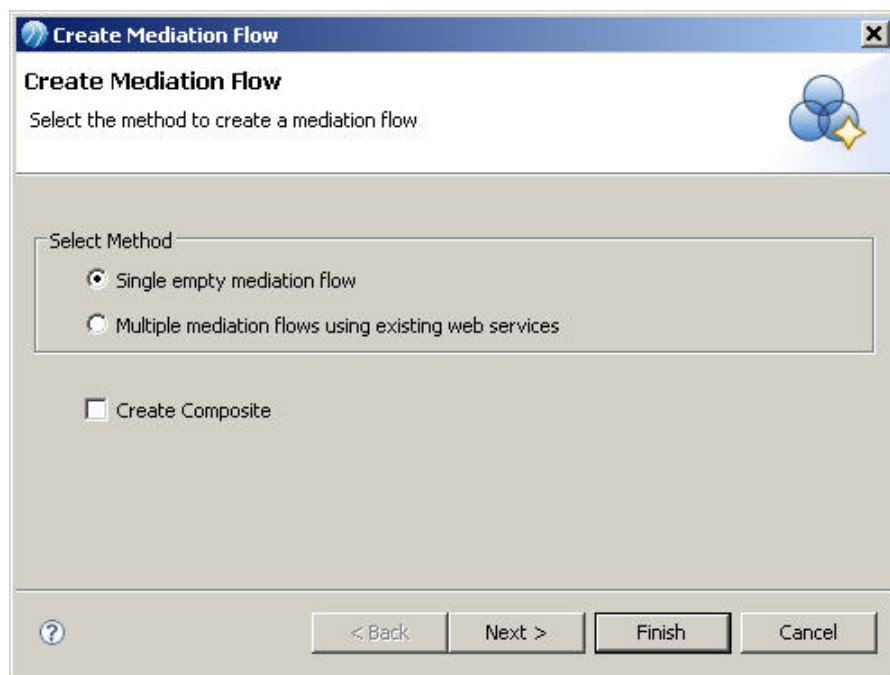
Starting the Mediation Flow Wizard

Use the following steps to start the mediation flow wizard.

Procedure

1. Right-click the Mediation Flows folder in the Project Explorer.
2. Choose **New > Mediation Flow** from the pop-up menu.
The Create Mediation Flow dialog opens.

The default option is to create an empty mediation flow.



Creating a New, Empty Mediation Flow

Creating a new, empty mediation flow enables you to start a mediation flow from scratch.

Prerequisites

You should at least have one WSDL file that describes the interface that you plan to mediate, but you can have zero or more target interfaces.

Procedure

1. Start the mediation flow wizard.
 - a) Right-click the Mediation Flows folder in the Project Explorer.

- b) Choose **New > Mediation Flow** from the pop-up menu.
The Create Mediation Flow dialog opens. The default option to create an empty mediation flow is active.

You can also select the Create Composite checkbox if you want to create a corresponding composite for this mediation flow. See [Mediation Components](#) for information about working with components and composites.

2. Click **Next**.
3. Supply a name in the **Mediation Flow Name** field.

If you chose to create a composite to correspond to the mediation flow, you can also name the composite in the Composite Name field.

You can also specify a different folder in the project for the mediation flow (and composite, if one is created).

4. Click **Finish**.
The **Mediation Flow Editor** opens. You can begin to add interfaces and configure your mediation flow.

Creating New Mediation Flows from Existing Web Services

To mediate existing web services, you can create new mediation flows for each interface. This is useful if there are multiple services and you want to create one mediation flow for each service.

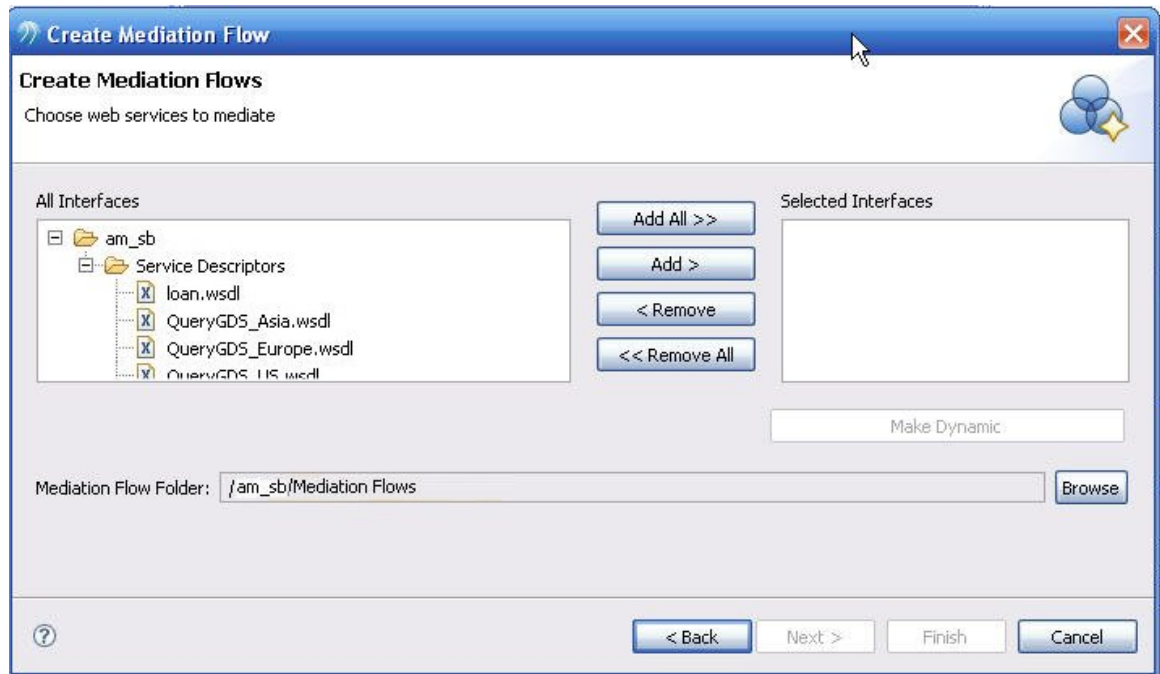
Prerequisites

Before creating mediation flows, ensure that the WSDL files that describe the interfaces have been imported into the project. See Composite Development for more information about importing WSDL files.

Procedure

1. Start the mediation flow wizard.
 - a) Right-click the Mediation Flows folder in the Project Explorer.
 - b) Choose **New > Mediation Flow** from the pop-up menu.
2. Select the option **Multiple Mediation Flows Using Existing Web Services**.
 - You can also select the **Create Composite** checkbox if you want to create a corresponding composite for the mediation flows.
See [Mediation Components](#) for more information about working with components and composites.

3. Click **Next** to select the interfaces for the mediation flow.



4. Select the WSDL files to use when you create mediation flows.
 - You can select and add WSDL files individually, or click the **Add All>>** button to add all files to the Selected Interfaces list.
 - You can remove one or more WSDL files using the **<Remove** and **<<Remove All** buttons.

A mediation flow is created for each WSDL file. The target interfaces and mediation interfaces are the same, and a path is automatically created between operations of the same name. If a WSDL file includes more than one port type, each port type is added to the mediation flow created for the file.

5. The next step depends on whether you checked the **Create Composite** option:
 - a) If you did not check the Create Composite option, click **Finish** to create the mediation flows and composite.
 - b) If you checked the option to create a composite, click **Next** to specify the binding type — JMS, SOAP/HTTP, SOAP/JMS.

If you select SOAP/HTTP, additionally specify the **Connector**.

Click **Finish** to create the mediation flows and composite.



The image shows a Windows-style dialog box titled "Create Mediation Flow". The subtitle is "Configure composite service binding details". In the top right corner, there is a logo consisting of three overlapping blue circles and a yellow star. The main area of the dialog is a light gray rectangle. Inside this area, the text "Service Binding/Transport:" is followed by three unchecked checkboxes: "JMS", "SOAP/HTTP", and "SOAP/JMS". To the right of these checkboxes is a text input field labeled "Connector:" containing the text "httpConnector". At the bottom of the dialog, there is a row of four buttons: a help button with a question mark icon, a "< Back" button, a "Next >" button, and a "Finish" button. A "Cancel" button is also present at the bottom right.

Create Mediation Flow

Configure composite service binding details

Service Binding/Transport:

- ☐ JMS
- ☐ SOAP/HTTP
- ☐ SOAP/JMS

Connector:

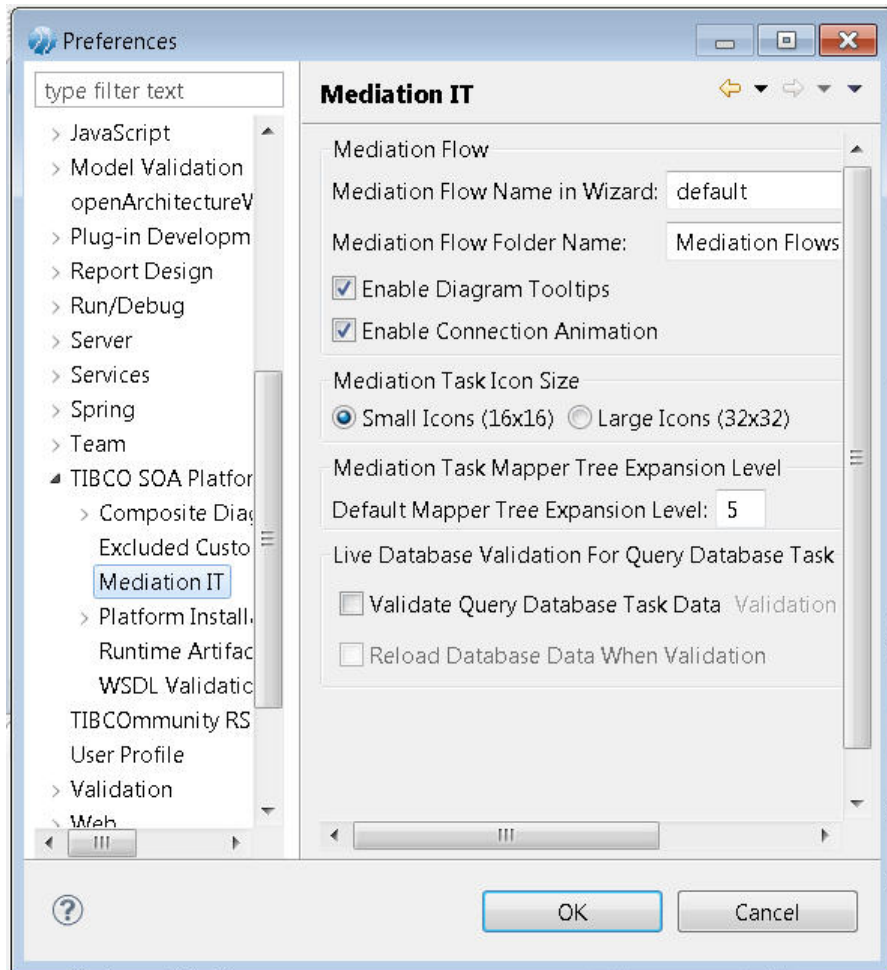
? < Back Next > Finish Cancel

Editing Mediation Flow Editor Preferences

You can set preferences for the Mediation Flow Editor.

Procedure

1. Select **Window > Preferences** to open the Preferences dialog.



2. Set values in the **Mediation Flow** section.
 - In the **Mediation Flow Name in Wizard** text box, provide the default name of mediation flows that you create with the **Single Empty Mediation Flow** option in the wizard.
 - In the **Mediation Flow Folder Name**, provide the name of the folder in which to store mediation flows.
 - Clear the **Enable Diagram Tooltips** check box if you want to disable the tooltips.
 - Clear the **Enable Connection Animation** check box if you want to disable animation.
3. Set values in the Mediation Task Icon Size section.
 - Check the **Small Icons (16x16)** radio button to display small icons in mediation flows and the palette.
 - Check the **Large Icons (32x32)** radio button to display icons in mediation flows and the palette.
4. In the **Mediation Task Mapper Tree Expansion Level** text box, type the default value for the depth you want to expand the left and right sides of the mapper.

5. Set values in the **Live Database Validation For Query Database Task** section.
 - Check the **Validate Query Database Task Data** checkbox to connect to the database during validation to determine if the Query Database task configuration is correct. The information being queried is the structure (tables and columns) of the database.
 - In the **Validation Timeout** field, provide the timeout (in seconds) for the validation task.
 - Check the **Reload Database Data When Validation** checkbox, if the database structure is changing, to query the database each time the validation process is run.

Working with Mediation Flow Properties

Mediation flow properties can be defined to store information such as values for current price markups, currency rates, or user names.

The properties are stored in the mediation exchange, and tasks in a mediation flow can use them. Properties are defined and removed using the Properties view of the mediation flow.

Validation of Message

When a mediation flow is created, a property called **VALIDATE_MESSAGE_DATA** is added by default.

At design time, a property **VALIDATE_MESSAGE_DATA** is defined in the mediation flow implementation. When set to true, this property is used to validate the incoming message. This property is accessed by the mediation component in the composite is exposed at the composite level as **MEDIATE_VALIDATE_MESSAGE_DATA**.

Having this property at the mediation component level allows for fine-grained control compared to defining it at the mediation implementation level.

A property **VALIDATE_MESSAGE_DATA** is available at the mediation implementation level when viewed using the Administrator UI. The default value of this property is false. Set this property to true to enable validation of message data received by the mediation component. Validation of message data happens for both the component service and reference.

Setting this property to true at the mediation implementation level enables validation on every mediation component on that particular node.

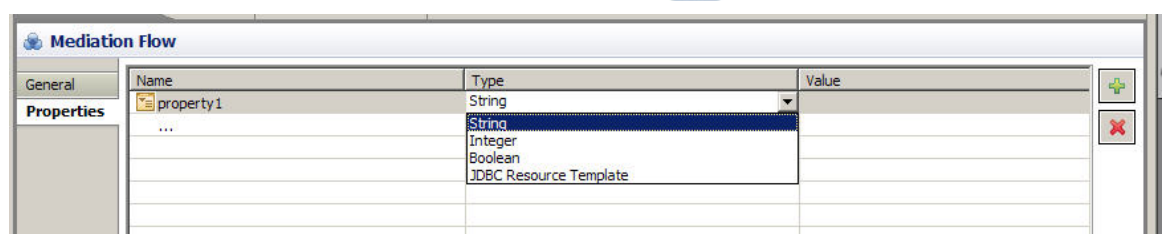
The value of the property set at the mediation component level takes precedence over the value set at the mediation implementation level. At runtime, when the incoming message (either a request message on the mediation interface or a reply message on the target interface) into mediation fails validation, an undeclared fault is returned to the consumer. The fault message will only indicate that a validation failure has occurred with no details provided. For more details about the cause of the validation failure will be contained in the log files.

Adding a Mediation Flow Property

You can define mediation flow property to store information.

Procedure

1. Click the ellipsis (...) in the **Name** field, or click the **Add**  button to the right of the table.



2. Specify a name, data type, and value for the property. You can select one of four property types:
 - String
 - Integer
 - Boolean
 - JDBC Resource Template

You can also specify properties when you create a mediation component. Component-level properties override the values of properties with the same name specified on the mediation flow.

Deleting a Mediation Flow Property

You can easily delete a mediation flow property.

Procedure

1. Select the property row in the table.
2. Press the **Delete** key or click the **Delete** button to the right of the table.

Working with Interfaces

Interfaces are collections of operations that WSDL files define.

WSDL files are typically contained in the Service Descriptors special folder in a project. You can obtain interfaces in a variety of ways, usually by importing WSDL files into a project or by using a UDDI registry service.

Composite Development describes the folders in an ActiveMatrix SOA project and how to obtain WSDL files and use UDDI registry services.

Adding Interfaces to Mediation Flows

There are many ways to add interfaces to a mediation flow, the method you use depends upon the requirements of your application.

Prerequisites


Before you add interfaces to your mediation flow, plan the needs of your application and determine which target and mediation interfaces you need. Some planning considerations are discussed in [Planning Target and Mediation Interfaces](#).

Procedure

- You can add interfaces to a mediation flow using one of the following choices:
 - If you plan to have a one-to-one relationship between target and mediation interfaces, you might use the technique described in [Creating New Mediation Flows From Existing Web Services](#).
 - If a WSDL file contains more than one port type, you can expand the WSDL file in the project tree and select only the interface you want to drag and drop into the mediation flow. You can also drag and drop the top-level WSDL file to add all interfaces within the WSDL file to the mediation flow.
 - If your target interfaces and mediation interfaces have different operations and schemas, drag and drop each interface from the Project Explorer into the appropriate area of the mediation flow.



Dragging and dropping an interface onto the mediation side of a flow creates an untargeted flow for each operation. When you select the mediation operation for an untargeted flow, the flow appears as a line ending in a question mark. You can then use a **Generate Reply**, **Throw Fault**, or **End Mediation** task on the flow without having to add a target interface. You can also drag and drop an untargeted flow to a target interface and mediate that interface.

- Another way to add an interface to the target or mediation interface list is to use the menu icon  at the top of the Mediation Interfaces area and Target Interfaces area of the mediation flow. The menu contains an Add Mediation Interface or Add Target Endpoint item, depending upon which side of the mediation flow you use. The Add menu opens a Select WSDL Port Type dialog where you can choose an interface to add.




You can add more than one copy of the same interface to the mediation interfaces side of the mediation flow. Doing so allows you to specify different business logic for the same interface. You can then expose each implementation of the interface to different clients.

For example, you could use this functionality to offer different qualities of service to different clients.

Deleting Interfaces from Mediation Flows

You can delete interfaces from either side of the mediation flow.

Procedure

1. Click the interface menu icon  in the title bar of each interface.
2. Select **Delete** from the pop-up menu.

Moving Mediation Interfaces

Interfaces can be moved within the target and mediation interface list.

Procedure

1. Click the header of the interface you want to move.
2. Drag the interface to the new location in the list of interfaces.

Using the AutoMediate Feature

The AutoMediate mechanism in the Mediation Flow Editor allows users to quickly add identical interfaces to both the target and mediation interface sides of the mediation flow with corresponding mediation paths between operations of the same name.

Procedure

- You can use AutoMediate using one of the following choices:
 - Drag and drop an interface to either the target or mediation interface area. Select the interface you have added to the mediation flow, then click and drag it to the opposite side of the mediation flow.
 - Drag and drop an interface to the target interface area. Then, click the projection icon in the title bar of the interface.
 - Drag and drop an interface onto the mediation paths area (the center area) of a mediation flow.

All mediation operations are connected to their corresponding target operations.



You can use the TIBCO AutoMediate Command Line tool to use existing services as input to create a fully functional composite application that generates a DAA that you can deploy into an ActiveMatrix runtime environment. See [TIBCO AutoMediate Command-Line Tool](#) for detailed information.

Creating Local WSDL Files

If you automatically create mediation interfaces, you may want to create local copies of the WSDL files. Creating local copies enables you to make changes to the copies without affecting other services or clients that use those WSDL files.

Procedure

1. In your mediation flow, locate the mediation interface you want to include in the local WSDL file.
2. Click the menu icon in the title bar of the interface and choose **Copy Interface** from the pop-up menu.

The Mediation Flow Editor creates a local WSDL file and places it in the same folder as the mediation flow. The name of the file is the same as the name of the mediation flow, with the file extension `.wsdl`.



Include additional mediation interfaces in the local WSDL file by repeating Step 2 for each interface.

Each interface you copy is placed into the same local WSDL file so that you can edit the file using the standard WSDL editor.

Supported Policies

Mediation interfaces supports intents.

Intents	Description
At least once	Specifies that the provider must receive every message sent to it by consumers at least once.
Transacted one way	<p>Specifies that references must send all out-only messages within a global transaction, and the ActiveMatrix framework must deliver the message only after the transaction commits.</p> <p>See Composite Development for more information on intents and policies.</p>

Working with Mediation Paths

Paths are created by dragging and dropping a mediation operation onto a target operation.

Paths can be automatically created in mediation flows, as described in [Adding Interfaces to Mediation Flows](#). Creating an input path also creates corresponding output and fault paths. You can click the Input, Output, and Fault icons at the top of the mediation paths area to view the corresponding path for each mediation operation.


The path for only one mediation operation appears in the mediation paths area. Select a mediation operation to view its path.

Mediation operations must have an input path. Typically, the input path leads to a target operation, but there can also be a route task that splits the path into more than one destination, or the path can lead to one of these tasks:


- A Throw Fault task

- A Generate Reply task
- An End Mediation task




If a mediation operation is not implemented — that is, if it does not have an input path — an error icon  appears in the bottom left corner of the operation's icon. You must implement all mediation operations in a mediation flow before deploying the project.

Changing Mediation Paths

On the input path, a small circle appears next to the directional arrow of the path . This allows you to change a path.

Procedure

- You can modify the target operation using the following choices:

Goal	Procedure
Move path to a different target operation	Click the circle and drag the path to a new location.
Move target operation on an output path	Click and drag the circle next to the target operation 

Result

The input and fault paths are automatically updated.

Deleting Mediation Paths

Mediation paths can be easily deleted.

Procedure

1. Select the path.
2. Press **Delete**, or right-click while hovering over the path and select **Delete** from the pop-up menu.

Route tasks create sub-paths and have some additional characteristics. See [Routing Messages in a Mediation Flow](#) for more information about working with route tasks.

Working with Message Context Properties

In a mediation exchange, the context of the message sent to the mediation operation includes information about the message transport (for example, HTTP headers or JMS message headers/properties), and security information.

TIBCO ActiveMatrix provides a way for the mediation flow to receive message context information and access its values in the mediation path. It also provides a mechanism for setting the message context data for the input message of the target operation and the output message of the mediation operation.

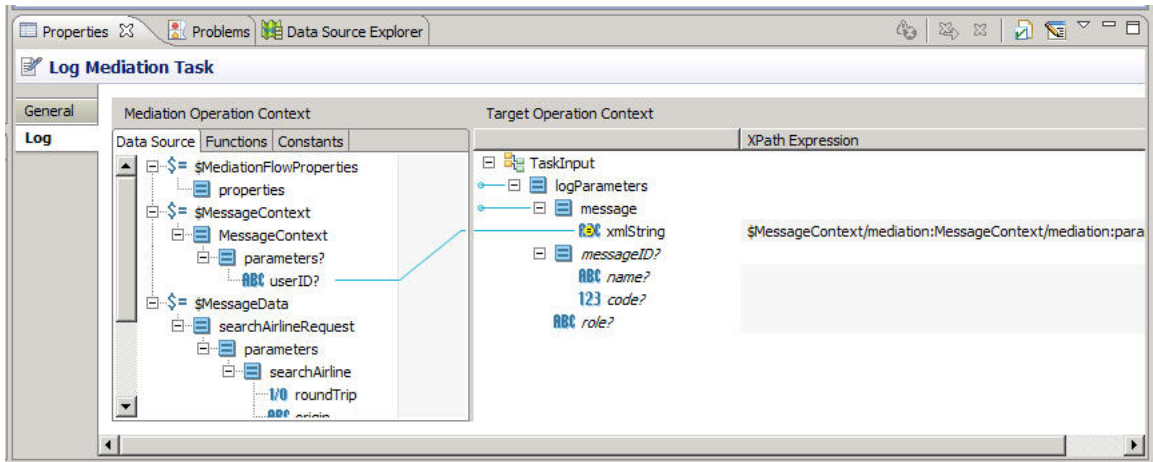
The Mediation Flow also allows you to configure an additional type of context parameter called **Mediation**. This type provides the security context, endpoint reference, and request message mechanisms.

Context Parameters

Context Parameters are variables stored the application session. These are useful when there is the need to share a parameter in several points in the application.

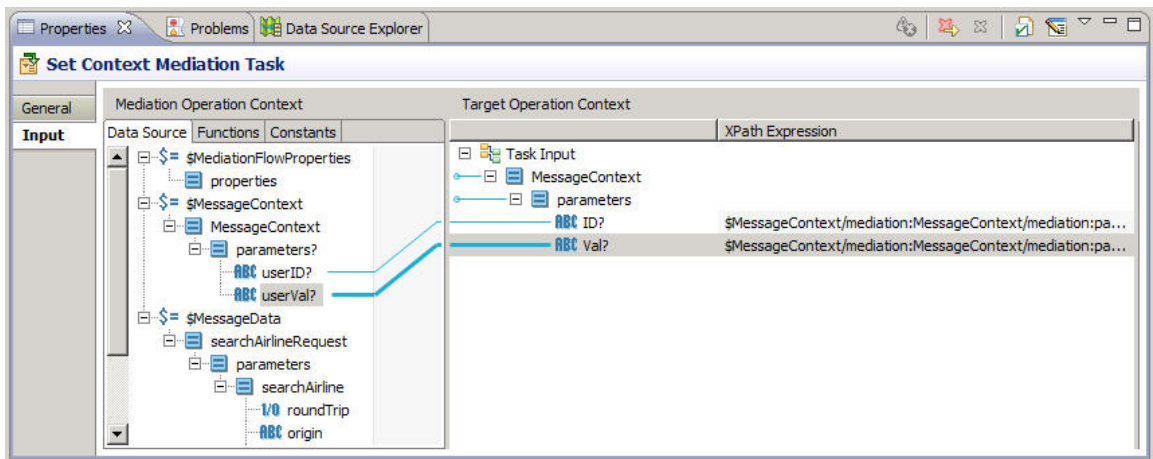
The context parameters available on the left side of the input mapper are contained under the root element MessageContext. Context parameters can be used like other elements in the tree.

The following figure shows the mapping of a context parameter, userID.



The Set Context mediation task is used to set values for the context parameters. The input mapper for the Set Context task shows the context parameters defined for the target interface or the mediation interface if the Set Context task is on the output or fault path.

The following figure shows the mapping of two context parameters. The parameters userID and userVal are defined for mediation interfaces, and the parameters ID and Val are defined the target interface.



Mediation Context Parameters

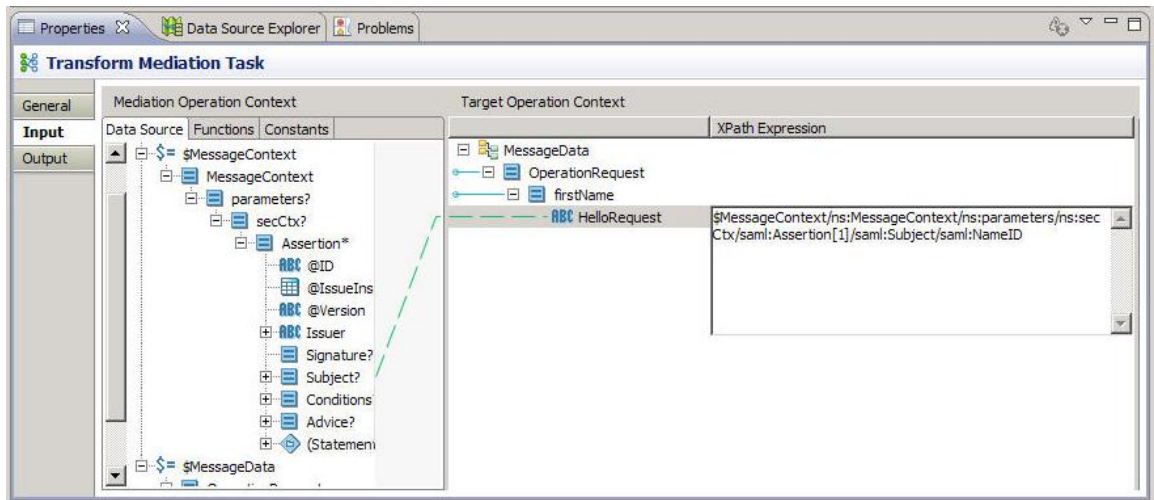
Mediation context parameter is available for the input direction only and provides functionality for security context.

The mediation component in the Composite Editor does not use these parameters. The values for these parameters are automatically passed to the mediation component and require no additional configuration.

Security Context

When a parameter of type Mediation and definition mechanism Security Context is added to the interface, the security context and the SAML assertion data is available for security context-based routing, transformations, or to log security context data.

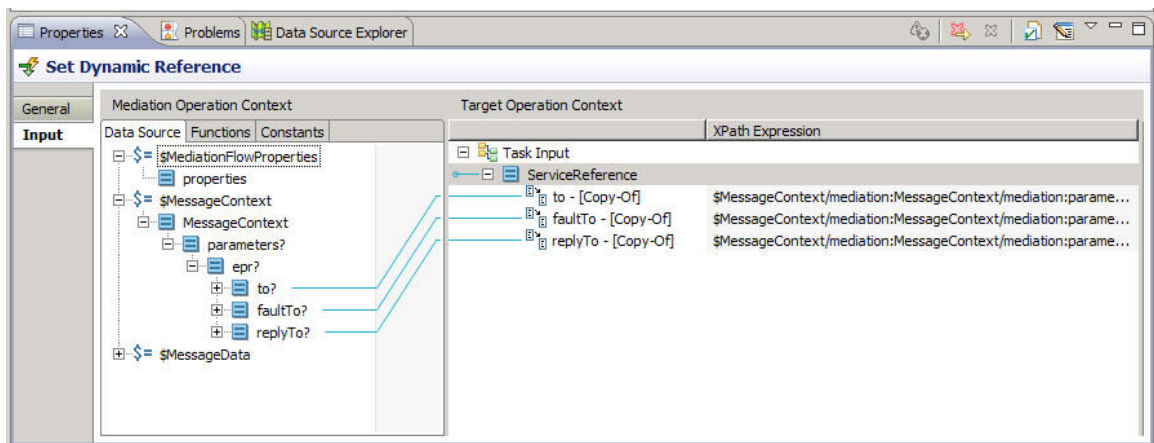
The following figure show a simple mapping of the SAML assertion data:



Endpoint Reference

A parameter of type Mediation and definition mechanism Endpoint Reference provides access to the endpoint reference schema for the mediation interface.

The Set Dynamic Reference task is used to provide the endpoint reference to the target invocation. The following screen shows one such mapping between the context parameters of the service and the reference.



The Endpoint Reference Mechanism of the Set Dynamic Reference task is set to WS-A Endpoint Reference. See [Set Dynamic Reference](#).

Request Message Context

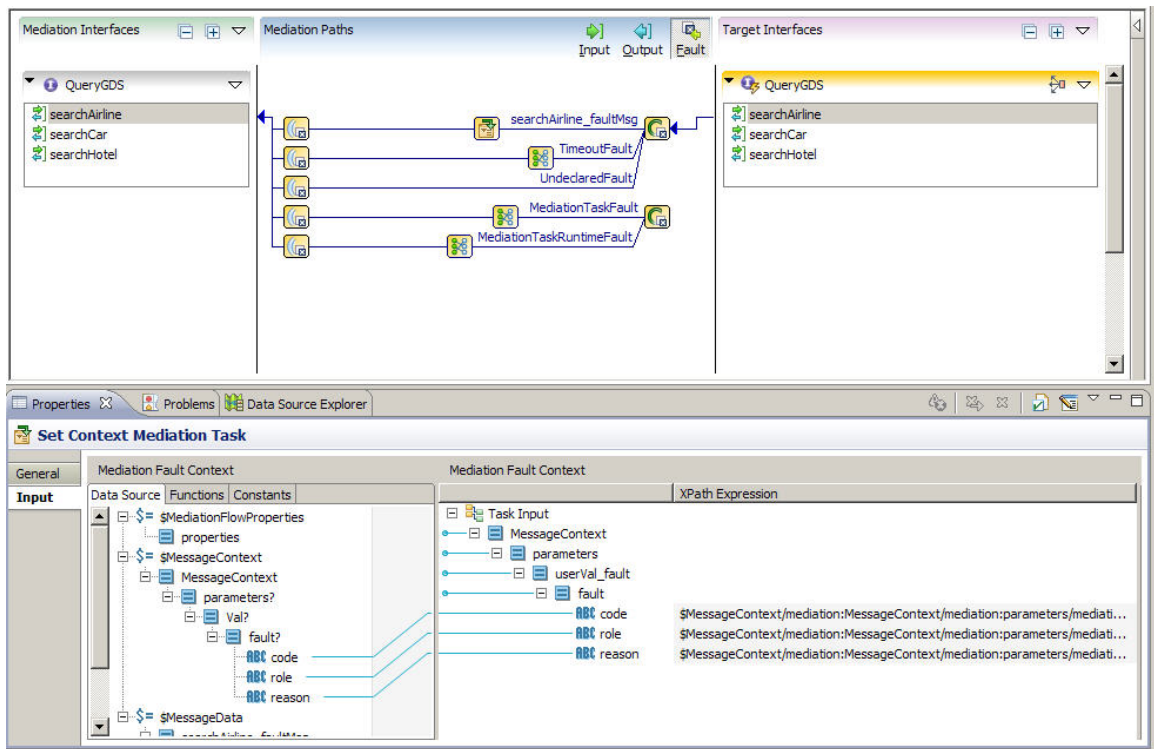
Provides access to the CorrelationID, ContextID, and ParentContextID parameters.

Undeclared Fault Headers

A context parameter of type Mediation and direction Fault is used to access the undeclared headers (code, role, and reason) provided by SOAP.

Using this type of context parameter requires the target interface to be bound to a SOAP endpoint, but no validation can be done to ensure that. The developer of the system just has to know that SOAP is being used.

The following screen shows the mapping between the context parameters on the Fault path of the Target interface and that of the Mediation interface.



Context Parameters in Mediation Components

Context parameters added to the interface or operation are propagated to the mediation component in the Composite Editor. The General tab of the Component Service and the Component Reference element has a section for context parameters.

All context parameters defined in the mediation flow will be exposed in the mediation component, except for the context parameters of type Mediation.


Context parameters can be added to the Component Service or Component Reference. These context parameters can then be pushed down to the implementation level. Context parameters of type Mediation cannot be added to the Component Service or Component Reference.

Adding Context Parameters

Context parameters can be configured at a mediation interface level, target interface level, or for an operation contained in the interfaces. Parameters added at the interface level are available for use by all the containing operations. Parameters added at the operation level can be used by those operations only.

Context parameters for the Mediation and Target interfaces are independent of each other. The Set Context mediation task is used to map values of the defined context parameters.

Procedure

1. Choose the interface or operation.
2. Select the **General** tab from the Properties view.
The **Operation Context Parameters** table is initially empty.
3. Click the  button located on the right side of the table.
4. Specify the parameter properties:

- Name: Name of the parameter.
- Direction: Choose between Input, Output, and Fault.
- Type: Choose between Basic, Message, Bag, Mediation.
- Definition: This is the definition mechanism.

If you chose Mediation as the context parameter type, the available definitions are Security Context, Endpoint Reference, and Request Message Context.


If the Direction is Fault and the Type is Mediation, the Definition mechanism is set to Undeclared Fault Context. See [Undeclared Fault Headers](#).

The context parameter is added to the chosen interface or operation.

Deleting Context Parameters

Context parameters can be deleted at either the interface level or the operation level.



Procedure


- To delete context parameters from the Mediation or Target interface, choose the context parameter and click  the button.
 - If the context parameter is deleted at the interface level, the parameter is deleted from all the operations.
 - If the context parameter is deleted from the operation level, the parameter is deleted from that operation only. If that operation was the last operation to which the context parameter was applied it is then removed from the interface as well.

Defining the Scope of Context Parameters

Parameters added at the interface level are available for use by all the containing operations. Parameters added at the operation level are only available for that specific operation.

Procedure

- You can define the scope of context parameter using the following options:
 - A parameter defined for operation A can be made available to operation B by selecting the parameter in operation B and clicking the  button.
 - A parameter defined for an operation can be made available to all other operations by selecting the parameter at the interface level and clicking the  button.

- The scope of a parameter used by operations A and B can be reduced by selecting the parameter in the operation where the parameter will not be used and clicking the  button. The operation will remain in the table but the tooltip will display the reduced scope.

Working with Exchange Variables

You can define an exchange variable for each mediation operation in your mediation flow. An exchange variable provides a location that stores data for use in all paths for a particular mediation operation.

For example, you might want to store a field from an incoming message, such as a correlation ID in a JMS header for the message. After it is stored, this data is available for all tasks in the input, output, or fault paths of a mediation operation.

Each mediation operation has one exchange variable. The exchange variable can have any structure. For example, the exchange variable can have repeating elements, if it is necessary to hold multiple instances of the same element.

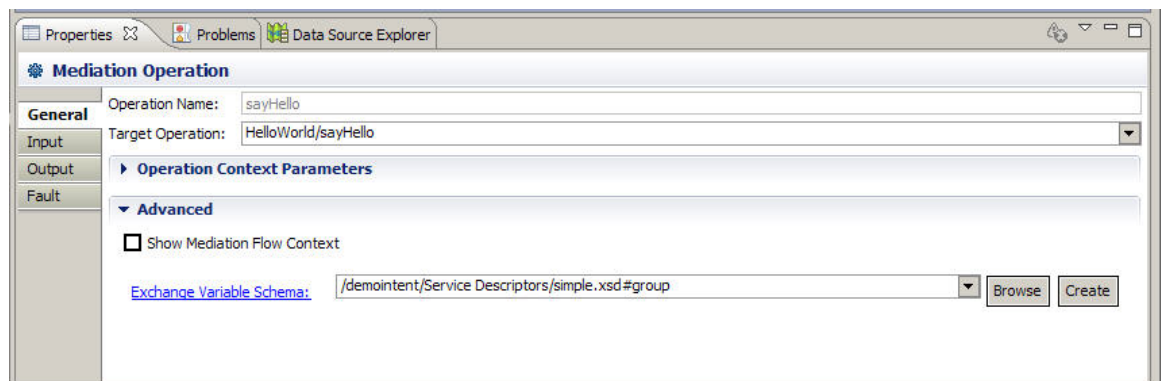
After it is defined, the exchange variable is available for all tasks that can access the mediation exchange in the input, output, and fault paths of your mediation flow. The values of the fields of the exchange variable are empty until they are set using the Set Exchange Variable mediation task. See [Setting the Exchange Variable](#).

Defining Exchange Variables

Exchange variables are defined in the Advanced section of the properties view of a mediation operation.

Procedure

1. Select a mediation operation in the mediation editor.
2. Expand the **Advanced** option from the **General** tab.



3. Specify a schema for the exchange variable in the **Exchange Variable Schema** field.
Set **Exchange Variable** supports only XSD elements, so the schema definition for the exchange variable must be stored in an XSD within your workspace.
You can use the following options to create an XSD (XML schema definition).
4. Use the following choices to create an XSD (XML schema definition).

Starting Point	Procedure
Browse button	Use the simple XSD editor with the Create button. See Creating Simple Schemas for information about the Simplified Schema Editor that opens when you click the Create button.

Starting Point	Procedure
TIBCO Business Studio	Use the XSD editor. See the Eclipse documentation, <i>XSD Developer Guide</i> , for more information about the XSD editor in TIBCO Business Studio.
XSD editor plug-in	Use your own plug-in.

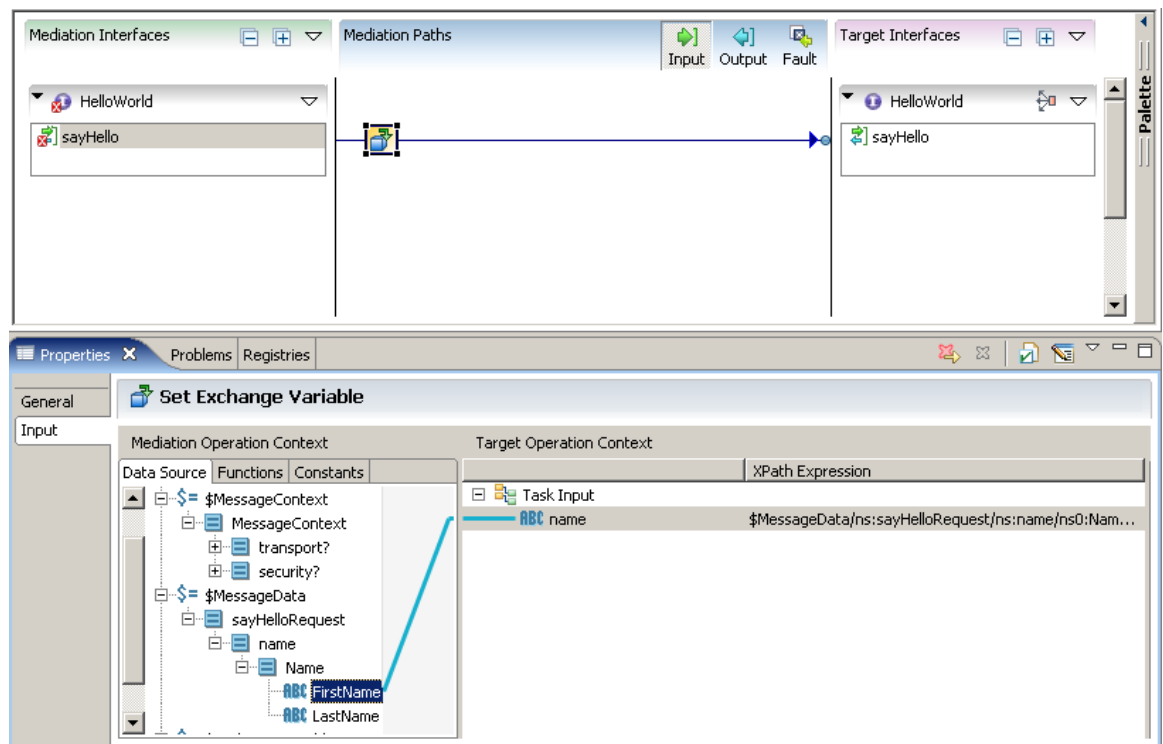
Setting Exchange Variable

The Set Exchange Variable task sets the value of the items within the exchange variable. The **Input** tab of the Set Exchange Variable task is a mapper panel that enables you to set any portion of the exchange variable for the currently selected operation.

Procedure

1. Use the **General** tab to specify a name and description for the task.

This tab is useful for providing documentation for tasks in your mediation flows.









2. Assign a name to the task, to identify the task in the mediation flow.
This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
3. Describe the task briefly.
This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
4. Click the **Input** tab.
This contains a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transform Tasks](#) for using a mapping panel.
5. To set the exchange variable, map values from the mediation exchange to the exchange variable element.



Creating Simple Schemas

You might need a schema for an exchange variable that is not stored in the project. The **Create** button in the **Exchange Schema** field opens a simplified schema editor dialog that you can use to create basic schemas. The simplified schema editor creates and stores the XSD file for the schema you create in the specified location in the project.

Procedure

1. Use the three fields of the simplified schema editor.
 - **Schema:** the structure of the schema. Use the buttons to add, move, and delete schema elements. Only elements can be created using this editor. If you must add attributes or create types, use the XSD editor in TIBCO Business Studio. Table 4 describes each of the buttons in the Schema field.
 - **Resource Name:** the name of the schema to create.
 - **Workspace Location:** the location in the workspace where the schema will be stored. Use the Browse button to locate a folder in another workspace.
2. Use the buttons for creating schema elements.

Button	Description
	<p>Adds a group to the schema. You can specify a name for the group and the type of group from one of these options:</p> <ul style="list-style-type: none"> • sequence in order—the elements in the group must appear in the order in which they are specified in the schema. • choice of one—the group is a choice group where only one of the elements in the group can appear at a time. • all in any order—all elements contained in the group can appear in any order.
	<p>Adds a complex element to the schema that can contain other elements. You can specify a name for the complex element, a type (from another stored schema), and the minimum and maximum number of occurrences of the element.</p>
	<p>Adds a primitive element to the schema. You can specify a name for the primitive element, a primitive type (string, integer, and so on), the minimum number of occurrences, and the maximum number of occurrences of the element.</p>
	<p>Adds a reference element. A reference element refers to a top-level element, allowing elements to be reused by reference. References in other schema resources are automatically maintained using imports.</p> <div style="border-left: 1px solid black; padding-left: 10px; margin-top: 10px;"> <p> The Simplified Schema Editor does not support duplicate namespaces. The assumption is that a given namespace is only imported once, and is associated with a single prefix.</p> </div>
	<p>Moves the currently selected element up one position in the schema.</p>

Button	Description
	Moves the currently selected element down one position in the schema.
	Deletes the currently selected element.

- After creating a schema, click OK to save the schema to the project.

Working with Tasks

Several operations are the same for all tasks within a mediation flow. Most operations can be undone by using the **Edit > Undo** menu item.

Details on how to work with each task in a mediation flow are described in the following topics:

- Invoking an Operation
- Logging Mediation Exchange Information
- Routing Messages in a Mediation Flow
- Transforming Data in a Mediation Exchange
- Replying to Messages
- Fault Processing in a Mediation Flow

See Reference , for information about any tasks not mentioned in the list above.

Adding a Task to a Path



Before you add tasks, expand the palette on the Mediation Flow Editor to show the list of tasks available.

Not all tasks can be added to all paths. See the description of each task in Chapter 12, Reference, for more information about the type of path where you can use the task.

Procedure

- You can add single tasks to a mediation path using the following options:

Operation	Procedure
Select	Select the task in the palette, and click the path line where you want to add the task.
Drag	Drag a task from the palette to a path. Hold the mouse button until the cursor is over the path.

- When a task can be added to the path, the path line becomes bold and the cursor changes to this icon  to indicate the task can be added.
 - If the task cannot be added to the path, the cursor changes to this icon .
 - If tasks are already on the current path, a vertical line appears on the path to show where the new task will be inserted. Move the cursor before or after the existing task to add the task to a specific location.
- You can add multiple tasks of the same type to a mediation path.
 - Select the task in the palette.
 - Press the **Ctrl** key while clicking on the path.

Route tasks cause the path to split into sub-paths. If your mediation flow requires routing, add the route tasks to the path first. Adding a route task to a specific location can be difficult when other tasks are already on the path.

Deleting a Task From a Path

Deleting a route task deletes all sub-paths and tasks after the route task.

You can delete Throw Fault and Generate Reply tasks, but the flow becomes untargeted. If you delete these tasks, you must retarget them as necessary.

Procedure

- To delete a tasks from a path use one of the options:

Option	Procedure
Mouse	Select the task in the path, right-click while hovering the cursor over the task, and choose Delete from the pop-up menu.
Keyboard	Select the task, and press the Delete key on your keyboard.

Mediation Components

Mediation flows provide implementations for mediation components.

Each mediation interface becomes a component service of the mediation component.

Each mediation flow property becomes a component property. You can override the values specified for mediation properties at the component or composite level.

Composite Development provides a complete description of composites and components and how they operate within the TIBCO ActiveMatrix architecture. You should be familiar with the procedures in that manual before attempting to work with mediation components.

You can create wires between composite services and component services, to provide bindings for mediation service consumers. You can also create wires between component references and composite references, to provide bindings to actual service providers for target interfaces. You can also create wires from other component services and references to and from the component services and references for a mediation component.

Invoking an Operation

The Invoke Operation task enables you to call an operation of any interface during processing of an input, output, or fault path. The Invoke activity can choose any operation from any interface in the target interface list.

For example, you can invoke an operation on the input mediation path and use the data in the reply message in subsequent tasks in the input path before the mediation flow invokes the specified target operation.

These examples describe use cases for the Invoke Operation task:

- Invoking a service to retrieve information, such as item price for a purchase order, a zip code for a city, or a shipping quote from a shipping service.
- Coordinating with non-automated processes, such as invoking a service to send an email message after the target operation returns a reply message.
- Basic orchestration with other services, such as invoking an approval service before invoking a target operation to allow a merchandise return.

An invoked operation can be either a one-way or request-response message exchange pattern.

If the invoked operation uses the request-response pattern, the mediation flow suspends execution until a reply is received from the invoked operation. The reply message from an invoked operation is placed in the mediation exchange in an element corresponding to the name of the Invoke Operation task. Subsequent tasks in the path can then access the reply message.

Fault Handling for Invoke Operations

Faults declared by an operation that an invoke activity references are caught and processed on the fault path. See [Fault Processing in a Mediation Flow](#).

Invoking Operations on Dynamic Interfaces

You can use the Invoke Operation task to invoke operations contained in a dynamic target interface. Dynamic target interfaces require a Set Dynamic Reference task that specifies the actual service to invoke. See [Dynamic Requests](#) for more information about dynamic target interfaces.

If the Invoke Operation receives a fault from the target service, the fault flow of the mediation is activated. The Catch Fault task has all target faults that might be generated by all of the invoke tasks, so you can mediate faults that are returned.

Configuring Invoke Operation Tasks

The operation to be invoked by the Invoke Operation task must be contained in an interface in the target interface list.

Input Tab

The Invoke Operation task requires an input message for the invoked operation. To construct the input message, use the **Input** tab of the **Invoke** tab. The **Input** tab is a mapper panel, similar to the mapper available in the Transform task. See [Transform Tasks](#) for more information about using the mapper panel.

Output Tab

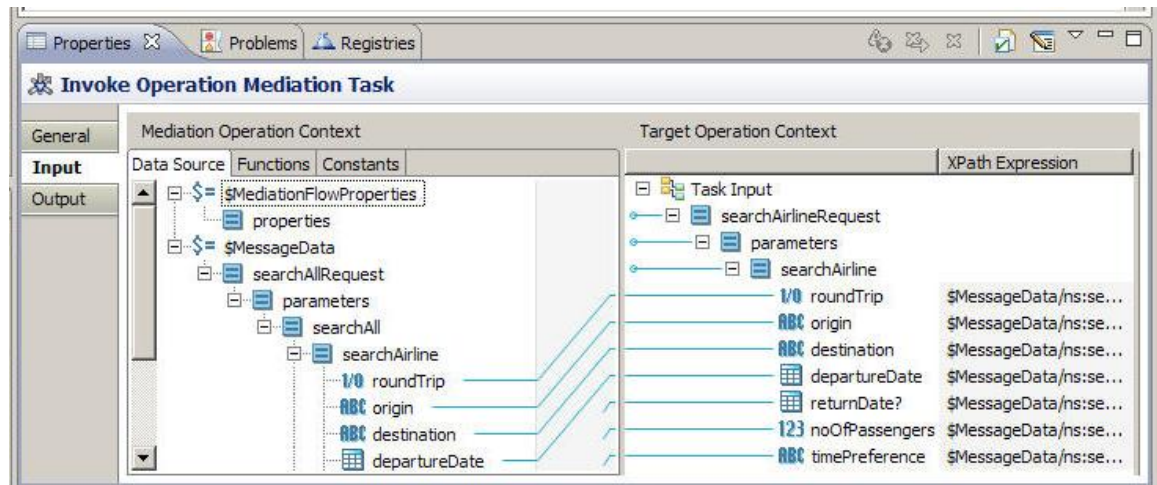
When the message exchange pattern of the invoked operation is request-reply, the **Output** tab displays a static schema tree to represent the output message of the invoked operation. If the message exchange pattern of the invoked operation is one-way, the **Output** tab of the **Invoke Operation Mediation Task** displays No Output Configured. The **Output** tab of the **Mediation Operation** is empty.

Procedure

1. Add an Invoke task to a path.
2. Select **Properties View > General tab > Target Operations**, and open the drop-down list to select an operation to invoke.

You can also press the Shift key and drag the task onto the operation in one of the target interfaces. (Dragging a task without using the Shift key rearranges the task on a path.)

When the Target Operation is selected for the Invoke Operation task, a green hint line appears to indicate which operation the task invokes.



Logging Mediation Exchange Information

Log tasks allow mediation flows to send data to a file (appender). By default, the appender for the mediation task is not configured.

You can place a log task on any input, output, or fault path. You can configure the log task to send any or all of these items to the log file.

- **Mediation flow properties** are the properties defined for the mediation flow. These properties are defined either on the **Properties** tab of the mediation flow, or in the composite or component containing the mediation flow. You can select all mediation flow properties, or you can select individual properties to log.
- **Mediation flow context** logs message context such as component name and mediation flow information, if the **Mediation Flow Context** option is set on the Advanced section of the mediation operation's **General** tab.
- **Message context** is information about the transport or security details of the message. See [Working with Message Context Properties](#) for information about the Mediation Flow Context option.
- **Message data** is the content of the message. Some tasks, such as custom tasks or Transform tasks, can change the content of the message. The Log task can be used to output the message content at any point in the mediation flow. You can use this information for debugging, auditing, or other purposes.
- **Contributed data** Mediation tasks, such as the Transform task or custom tasks, can add or contribute data to the mediation exchange. For example, the Log task can be used to output any data that previously executed mediation tasks contributed to the mediation exchange.
- **Exchange Variable** If the mediation operation has an exchange variable set, the exchange variable appears as one of the items to log. If Log All Items is selected, the Exchange Variable is automatically logged if it is used.

Mediation Appenders and Loggers

By default, the runtime informational (INFO), warning (WARN) and error (ERROR) messages logged by the mediation component or the mediation log tasks are sent to the log file of the ActiveMatrix node or the associated appender.

Using the TIBCO ActiveMatrix Administrator to specify a unique appender for the mediation component or the mediation log tasks is also possible. The Administrator allows the user to configure logger and corresponding appenders at application or component level. Refer to *Administration Guide* for more details on application or component logger configuration.

Two mediation loggers are available:

- `com.tibco.amx.it.mediation`
The logger named `com.tibco.amx.it.mediation` is used by the mediation component to log runtime error, warning, informational or debug messages and must be applied to the node where the mediation application is running.
- `com.tibco.amx.it.mediation.logTask`
The logger named `com.tibco.amx.it.mediation.logTask` is used by the mediation log tasks and must be applied to the mediation application. This logger is available only if it has been configured at design time.

To send the data logged by the mediation log tasks to a specific appender, either one of the two logger names can be used in the Administrator to configure the application or component level loggers.

However, to isolate the data logged by the mediation log tasks from rest of the mediation component messages, the logger named `com.tibco.amx.it.mediation.logTask` must be used.



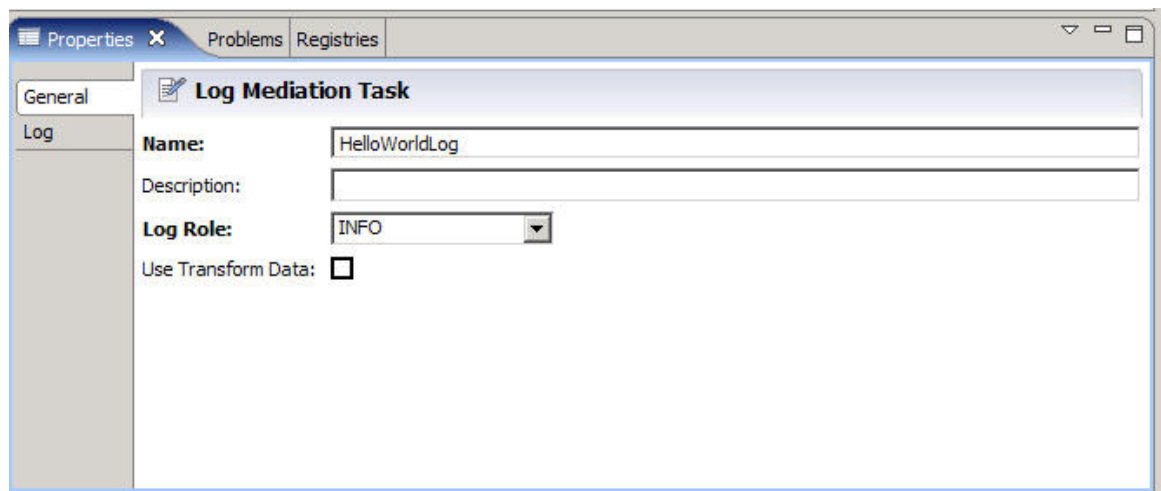
By default only the informational (INFO), warning (WARN) and error (ERROR) messages are written to the log file of the node or the associated appender. The Mediation component or mediation log task's debug (DEBUG) or trace (TRACE) messages are not written to the log file of the node. To view debug or trace level messages, a logging appender must be configured at a debug level.

Configuring a Log Task

After adding a log task to a path within a mediation flow, specify the type of information you want to log. The **Log** tab of the Log task configures the information to send to the log file.

Procedure

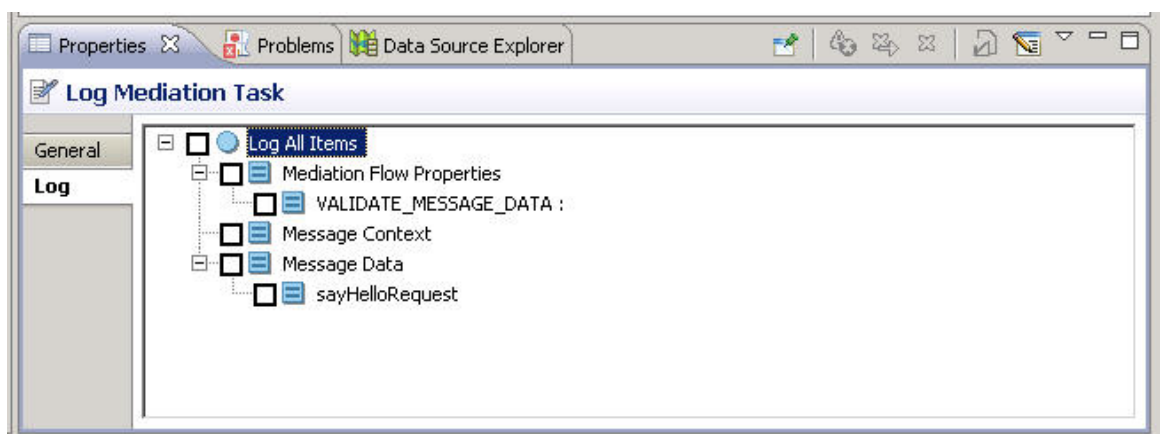
1. Give the task a name and description.



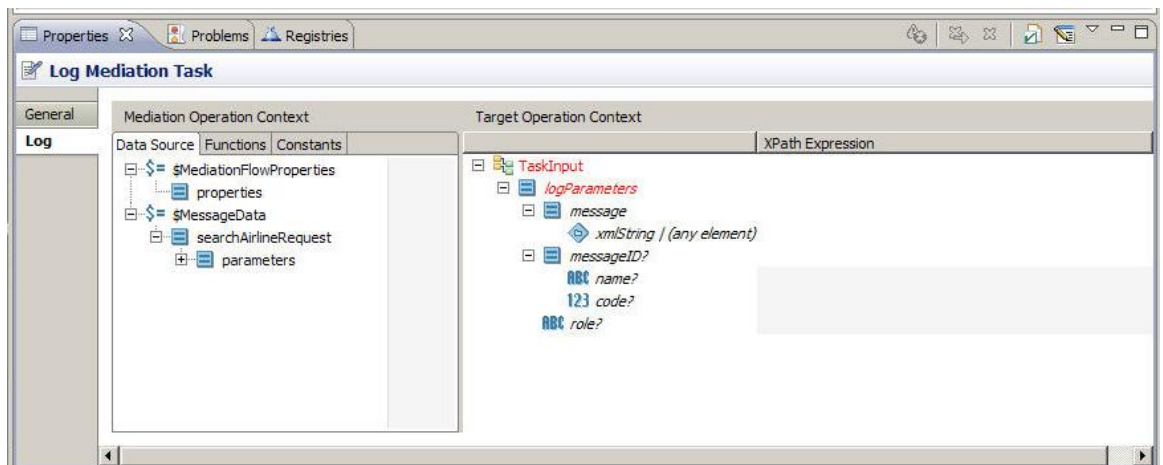
2. Choose the role level of logging for the task.
3. Set the Use Transform Data option.

This option controls the appearance of the Log tab, where you configure the information to send to the log file.

- If **Use Transform Data** is not selected (the default) on the General tab of the Log Task, the Log tab appears and shows top-level message information, and you select the items to send to the log file. To send all available information to the log file, select the **Log All Items** box.

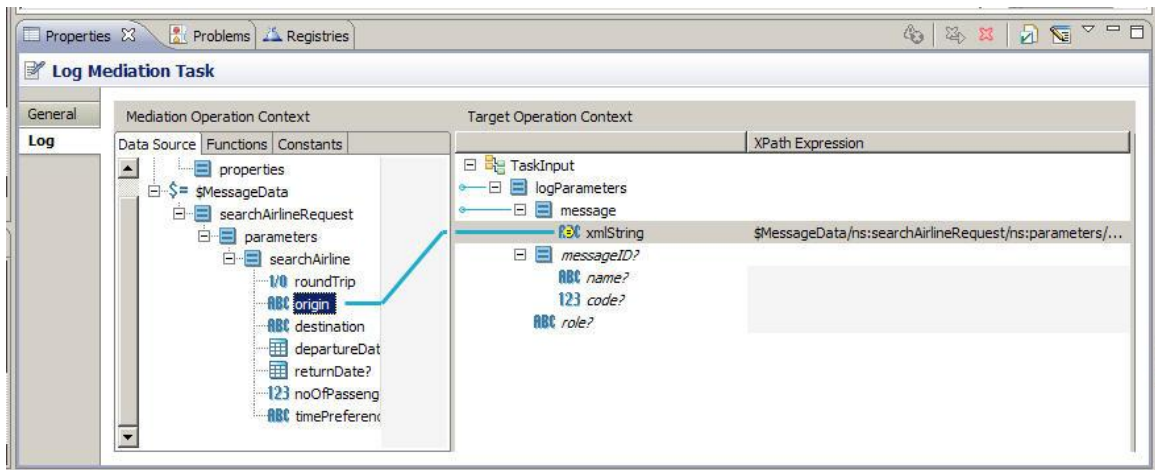


- If the **Use Transform Data** option is selected on the **General** tab of the Log task, the Log tab appears as a mapping panel. You can use this panel to build custom log messages.



Message mapped to the message element

This example shows part of a message mapped to the message element:



The messageID element is useful if you need to specify a code, or map from a code that is included in a message. See [Information for Custom Log Messages](#) for detailed information about the messageID element.

You can set a property on the mediation flow for the Log task to use at run-time, to override the **Log Role** setting in the **General** tab if they are different. For example, you might set the **Log Role** to debug during development, but set the run-time property to info.

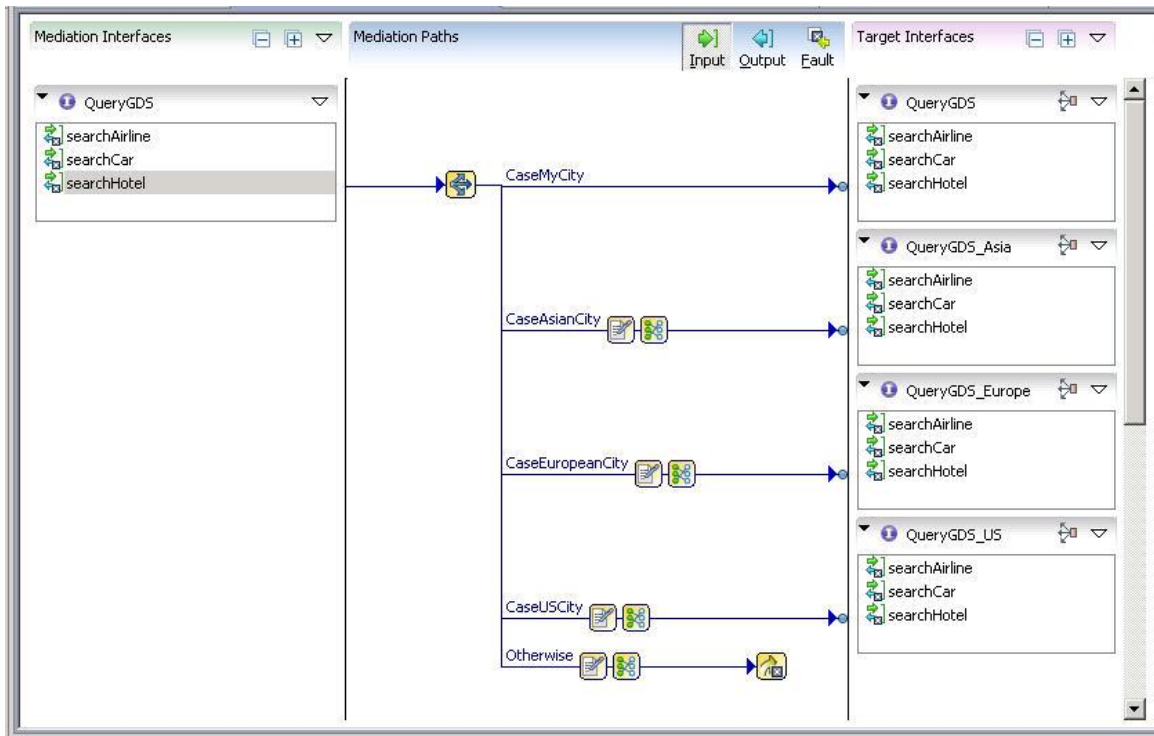
See [Transform Tasks](#) for more information about using a mapping panel.

Routing Messages in a Mediation Flow

Route tasks are used to specify that messages can be delivered to different destinations based on values within the message data or within other data in the mediation exchange, such as the security context.

Route tasks enable you to send messages to a specific destination based on conditions that you specify. Data from the mediation exchange, such as the message context or the message body, can be used to construct the routing conditions. For example, you might route incoming messages to a local server from 9:00 a.m. to 5:00 p.m., but outside of those times, route incoming messages to a different server.

An example of using Route Task



The example shows the input path of a system that searches for travel reservations. For the `searchHotel` mediation operation, incoming messages are routed to the appropriate service, based on the city specified in the search request:

- If the city is that of the requestor, the message is sent to the `QueryGDS` service.
- If the city is in Asia, the message is sent to the `QueryGDS_Asia` service.
- If the city is in Europe, the message is sent to the `QueryGDS_Europe` service.
- If the city is in the United States, the message is sent to the `QueryGDS_US` service.
- A fault is thrown if the city is not the requestor city, or in Asia, Europe, or the United States.

The two types of route tasks are, Route and XPath Route.

- **Route** task enables you to define basic route conditions.
- **XPath Route** task allows more flexibility in the expressions you can use to define a route condition.

The type of condition that you must define determines which route task is appropriate for your application. See [Routing Conditions](#).



If you create Route tasks and later decide that a more complex routing condition is required, you can easily convert the Route task to an XPath Route task. See [Changing Route Tasks to XPath Route Tasks](#).

Paths and Route Tasks

Route tasks can be added only to input paths.

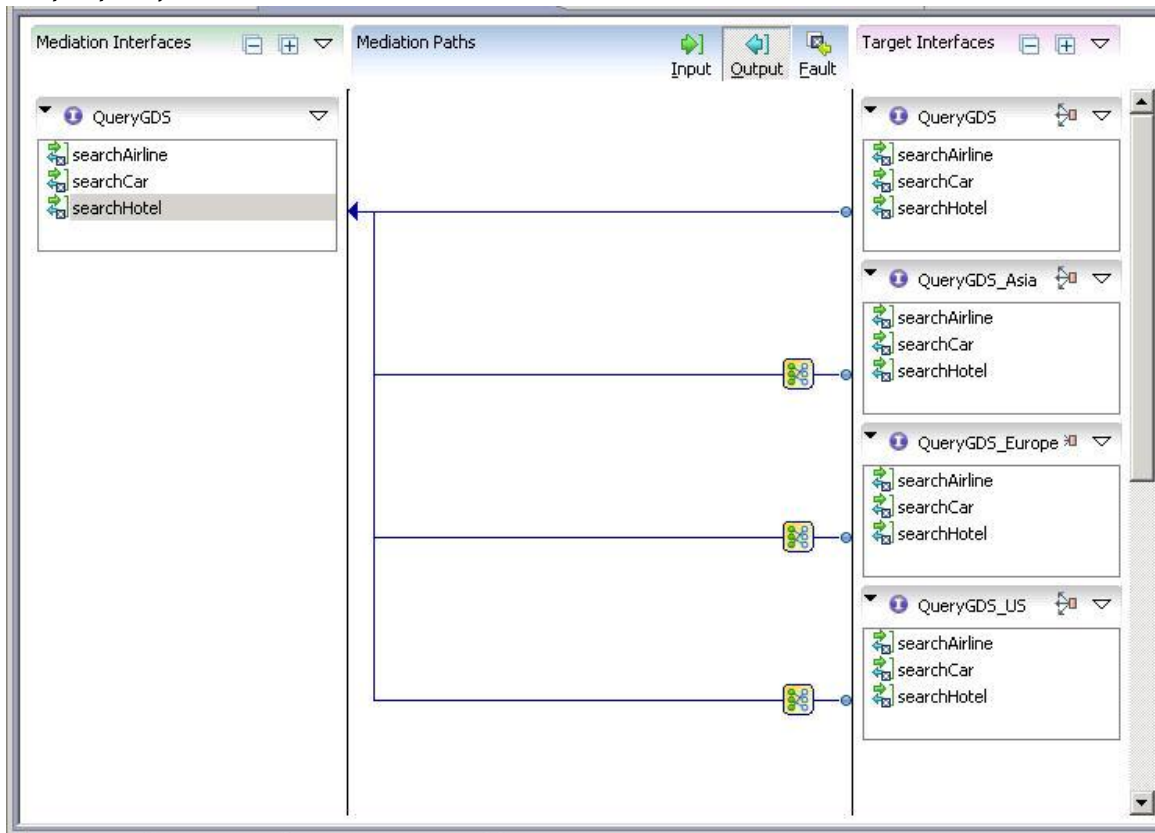
Route tasks send each incoming message to a single destination based on which route case evaluates to true, or to a single destination designated as otherwise if none of the cases evaluate to true.

Paths on the input flow to a target operation correspond to paths on the output and fault flow from that target operation. Paths ending in Throw Fault have a corresponding mediation fault path on the Fault flow. Paths ending in a Generate Reply task have a single, common Handle Reply path on the Output flow.



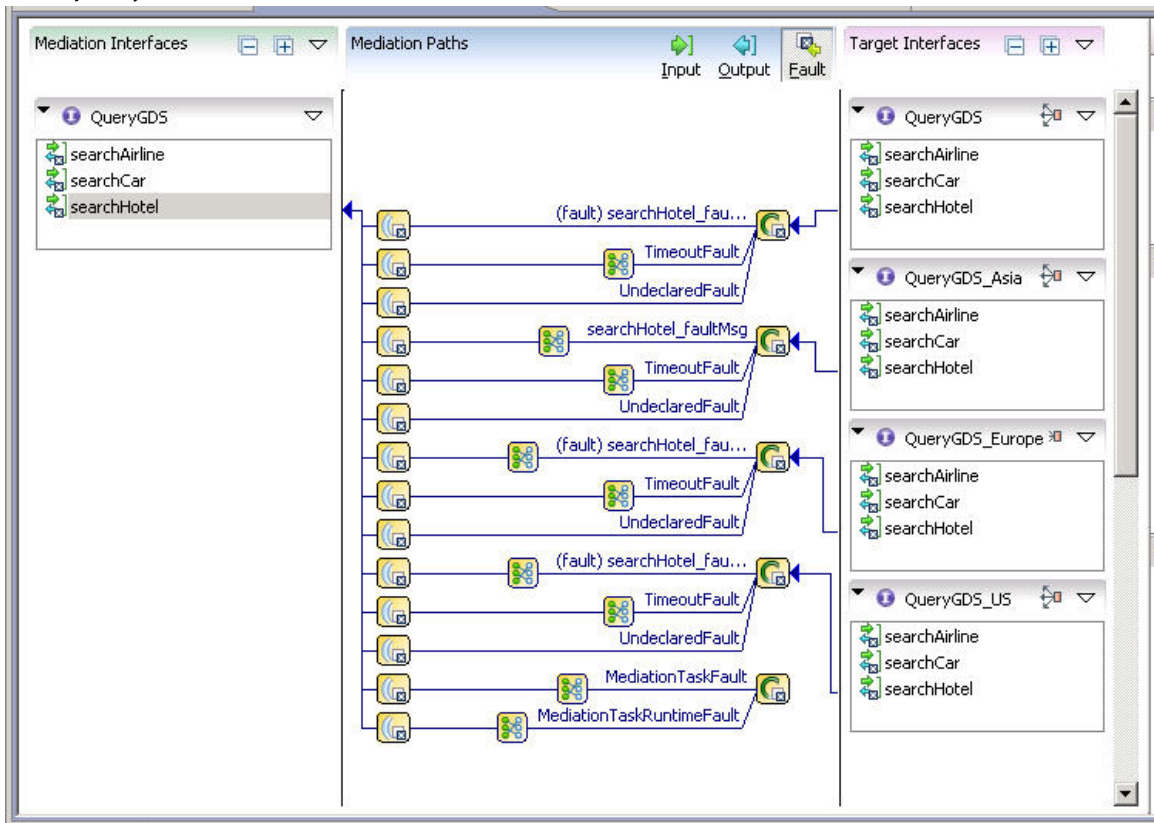
You can only introduce the route in the input path, and the response (output or fault) always returns to the original requester—the requester that invoked the mediation operation.

Output path for a route



Mediation tasks can be added to sub-paths after a route activity. Typically, you use a Transform task when the input, output, or fault message schema does not match the mediation operation message schema.

Fault path for a route



Defining a Route

The steps for defining a route task are the same regardless of the type of route task you are using.

Prerequisites

Before creating a route, your mediation operation must contain the mediation interface and one or more target interfaces that contain operations between which you route messages.

Procedure

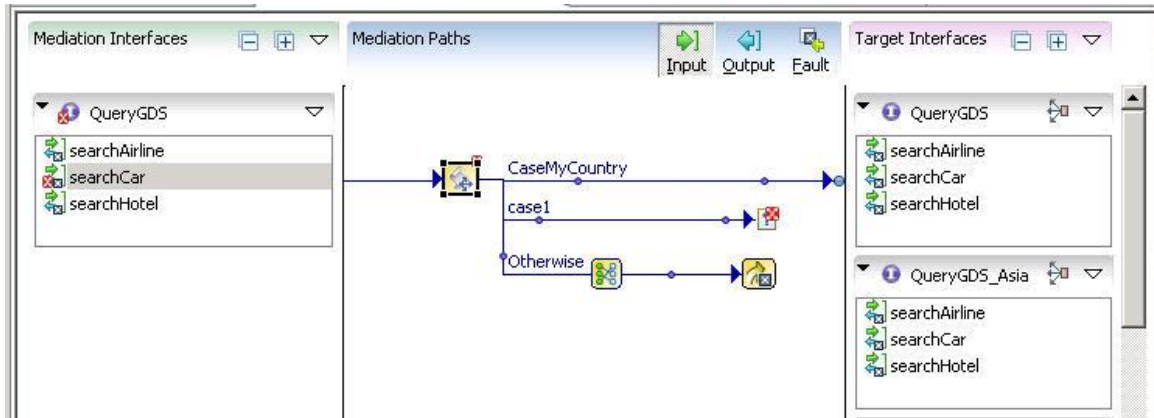
1. Add a Route or XPath Route task to an existing input path. If you have not yet created any input paths in your mediation flow, draw the path between your mediation operation and one of the target operations that you want for the destination for the route.
After a route task is added to an input path, a default case and an Otherwise case are created. Cases are the conditions that are evaluated to determine which sub-path a message takes. The Otherwise case is always present, and is used when all other cases evaluate to false.
 - Create more cases for the route, to create sub-paths to other target operations in your mediation flow.
2. Add variables to hold the value of data from the message content or message context. These variables are used in the routing conditions you specify in each routing case.
3. Specify the routing conditions for each case, using the variables that you have defined for the route.
4. Use the **Input** tab on the route task to map data from the message context or message content to the variables you defined in step 2.

Adding Routing Cases

Routing cases define the potential destination for the route. Each case leads to a different potential target operation. You must specify two things for each routing case: a name for the case and the destination to which the case leads.


Each routing case leads to one potential target operation. Target operations cannot be shared among routing cases. The relationship between routing cases and target operations must be one-to-one.

Adding a routing case



Procedure

- Use one of the following starting points to define a routing case.

Starting Point	Procedure
Input task	<ol style="list-style-type: none"> Click the route task in the input path of the mediation flow. Drag the cursor to the destination. <p>The sub-path is automatically drawn, and a case with a default name is added to the Decision tab.</p>
Decision tab	<ol style="list-style-type: none"> Open the Decision tab of the route task. Click the Add Case button  on the toolbar. <p>The Add Case button creates a case, but it does not lead to a specific target operation. The image shows the sub-path that is displayed in this situation. The sub-path leads to an error icon.</p> <p>See Modifying Case Names or Destinations for details on how to change the destination of the routing case to a valid target operation.</p>

Specifying Case Targets in the Decision Table

You can specify a target in the route task Decision table. If you retarget a Route task, the entire nested routing structure is replaced.

Procedure

- Click inside the cell where the target is located.
- Choose a target from the drop-down list.

- Target operations that are not already targeted.
- Generate Reply or Throw Fault mediation tasks.
- End Mediation task for one-way (in-only) operations.
- Route and XPath Route mediation tasks, which enables you to build nested routing structures.

Modifying Case Names

Use the **Decision** tab on the route task to change the name of the routing case.

Procedure

1. Click the name in the **Case** column.
2. Edit the name in the text box.

Modifying Destinations

Use the **Decision** tab on the route task to change the destination of the routing case.



Procedure

1. Click the name in the **Case** column.
2. Select the option that matches your goal.
 - For selecting a new target operation, use the drop-down list in the **Target Service/Operation** field on the **Decision** tab to specify the new target operation.
 - For newly created routing cases that point to an error icon, click the error icon and drag the cursor to the target operation.
 - For routing cases that point to a valid target operation, click the round ball at the end of the input path, and drag the cursor to the target operation.

Moving Cases in the List

Cases are evaluated in the order in which they appear in the list. The first case whose condition evaluates to true is taken, so you might need to move cases up or down in the list.

Procedure


1. Select the row of the case.
2. Click the **Move Up**  icon on the toolbar to move a case up on the list.
 - Click the **Move Down**  icon on the toolbar to move a case down on the list.

Deleting Cases

You can delete a routing case in two ways.

Procedure

-



Starting Point	Procedure
Mediation flow	<ol style="list-style-type: none"> 1. Select the sub-path of the routing case in the mediation flow. 2. Right-click and select Delete from the menu, or press Delete.
Route task	<ol style="list-style-type: none"> 1. Select the row of the routing case on the Decision tab of the route task. 2. Click the Delete Case icon  on the toolbar.

Nesting Multiple Route Tasks

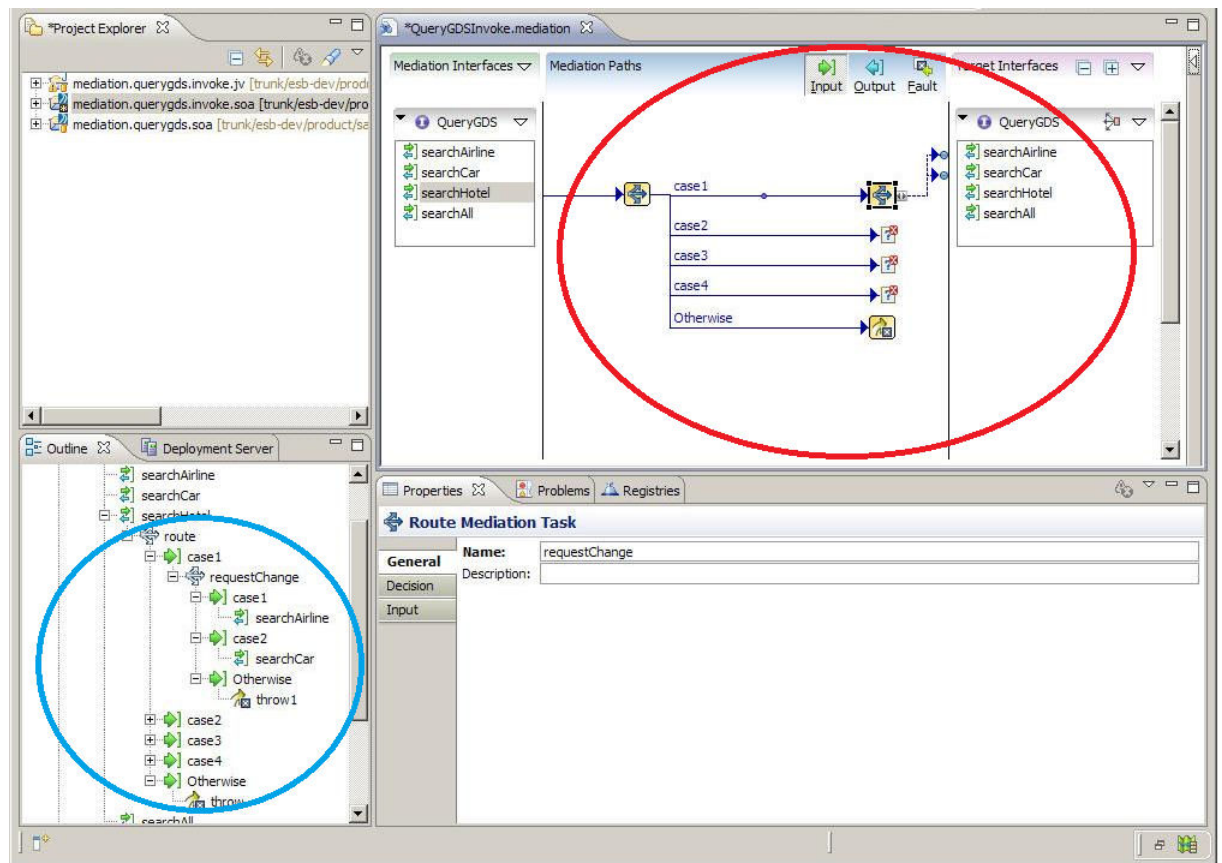
You can use multiple, nested Route tasks to send a message to a target operation. Doing so enables you to create complex mediation paths with multiple conditions.

When you use nested Route tasks, the mediation path shows whether a route goes directly to a target operation, or goes through another Route task first. When multiple Route tasks are in the mediation path, the task output details (or case paths) from only one of the Route tasks is visible at a time.

Procedure

1. To see the output details of another Route task in the mediation path, click the  button next to the Route task icon, or use the outline view to navigate to a specific Route task.
2. To see the level of a route in a nested set of Route tasks, place the cursor over the  button next to the Route task icon.

In this example image, the top-level Route task shows nested routes in a mediation flow in the red circle. Nested routes appear in the Outline view in the blue circle.



Adding and Deleting Variables


Variables hold the data that you use in expressions for each routing case. Variables are managed on the Decision tab of the route task.




After a variable is created, you cannot change its name or data type. To change a variable, you must delete the variable and create a new one.

Procedure

- You can add or delete a variable.

Task	Procedure
Add a variable	<p>Click the Add Variable icon  on the toolbar of the Decision tab. These data types are available:</p> <ul style="list-style-type: none"> String (default) Integer Boolean Float Double

Task	Procedure
	<ul style="list-style-type: none"> Decimal
Delete a variable	Click the Delete Variable  icon on the toolbar.

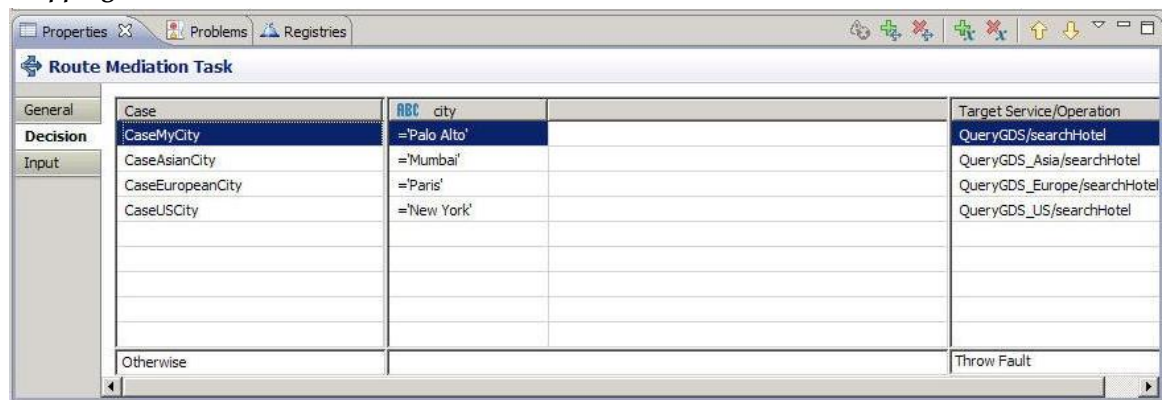
Mapping Data to Variables

After creating a variable, you must map data from the mediation operation to provide a value for the variable.

Procedure

- Click the **Input** tab of the route task.
In the **Input** tab of a route task, the right-hand panel is labeled **Rule Variables**. The schema in the right-hand panel contains a list of the variables that you have defined for the route task.
- Use XPath expressions to provide a value for each variable in this schema.
In the example shown, one variable named city is used to determine the destination of the message. In this example, the city specified in the search request is mapped to the variable named city. The value of the variable is then used in routing conditions to determine which target operation should receive the message.

Mapping values to variables



The section [Schema Components](#) explains that schema components on the left side of the mapper are not validated against the message schema, and their data types are thus not guaranteed. Therefore, data used within XPath expressions on the right side of the mapper is treated as untyped strings. Simple drag-and-drop mappings are not affected. However, if you want to perform data type-dependent comparisons or operations, you must use the Constructor Functions on the Functions tab (for example, `xsd:int()`) to correctly specify the data type. For example, to add two integers, the XPath expression would be:

```
xsd:int($MessageData/int1) + xsd:int($MessageData/int2)
```

Routing Conditions

Routing conditions determine which sub-path a message takes. Routing conditions are specified in order, and the message is sent along the sub-path corresponding to the first condition that evaluates to true. The Decision tab of the route tab contains the routing conditions for the route.

Routing conditions are XPath expressions, but each type of route task has a different method of specifying routing conditions:

- The Route task enables you to specify basic comparison expressions for each variable you have defined.
- The XPath Route task enables you to use more complex XPath expressions.

The type of route you use depends on the complexity of the routing conditions you need to define.

Conditions for Route Tasks

Route tasks create a simple comparison condition for each variable you have defined. A Route task is useful in situations where a basic comparison of a few variables can be specified.

Routing conditions for Route task

Case	RBC city	Target Service/Operation
CaseMyCity	=Palo Alto	QueryGDS/searchHotel
CaseAsianCity	=Mumbai	QueryGDS_Asia/searchHotel
CaseEuropeanCity	=Paris	QueryGDS_Europe/searchHotel
CaseUSCity	=New York	QueryGDS_US/searchHotel
Otherwise		Throw Fault

In this example, basic equality comparisons are performed for each case. When the *city* variable is equal to *Palo Alto*, the case named *CaseMyCity* evaluates to true, and its corresponding sub-path is taken. If none of the routing conditions evaluate to true, the sub-path for the *Otherwise* case is taken.

Editing Route Task Conditions

The Route task is useful in some situations where a basic comparison of a few variables can be specified.

Procedure

1. Click the cell for each variable and each case.
2. Specify a comparison operator and a constant value for comparison.
Basic comparison operators are available in a drop-down list in the condition:

- = (equal)
- != (not equal)
- < (less than)
- <= (less than or equal)
- > (greater than)
- >= (greater than or equal)
- =true() (only for variables of type boolean)
- =false() (only for variables of type boolean)

All conditions for each case must evaluate to true for the condition to be true.

In the example loan application shown, the operation SimpleLoanPortType/SimpleRequestLoan can be used in two circumstances:

- For loan amounts that are less than or equal to \$50,000.
- When the applicant has a credit score above 700.

If neither of these conditions is true, the LoanPortType/RequestLoan operation that requires more information from the applicant must be used.

Routing with more than one variable

Case	Amount	CreditScore	Target Service/Operation
SimpleLoanRequest	<=50000	>700	SimpleLoanPortType/SimpleRequestLoan
LoanRequest	>0		LoanPortType/RequestLoan
Otherwise			ThrowFault

Conditions for XPath Route Tasks

XPath Route tasks allow more complex comparisons for each case than does a Route task. You can specify an XPath expression for each case that examines the value of one or more of the variables that you have defined.

For example, you can create an expression that specifies a range of time (such as 9AM to 5PM), or you can create an expression that compares two or more variables. Your expressions are not limited to simple comparisons, and you do not need to use any of the variables you have defined in the expressions.

XPath Route tasks are more flexible than Route tasks, but specifying the expression is more complex. You must type the XPath expression in the condition field next to each routing case.

Variables are referenced in the XPath expressions for each routing case by their names. Unlike XPath expressions in the Transform task, you do not need to use a dollar sign to specify the root of the path to the variable. For example, the expression to determine if the city variable is equal to *Palo Alto* would be:

city = "Palo Alto"

The Transform task has a graphical XPath editor that you can use as a reference for creating XPath functions for the route task. See [Using XPath](#) and [Data Function Tabs](#) for more information about XPath.



As noted in [Mapping Data to Variables](#), data type-dependent comparisons and operations should use constructor functions to typecast the data.

Changing Route Tasks to XPath Route Tasks

You can change a Route task to an XPath Route task if, for example, you originally create Route tasks then later realize that a more complex routing condition is required. Instead of removing the existing Route task, you can convert a Route task into an XPath Route task.



Converting a Route task to an XPath Route task is a one-way operation. You cannot convert an XPath Route task to a Route task.

Procedure

1. Select the Route task to convert in a mediation flow diagram.
2. Right-click the Route task and select **Convert to XPath Route** from the menu.
All variables and cases are maintained, and routing conditions are converted to the correct XPath syntax.
3. Change the condition for each case, as necessary.

Transforming Tasks

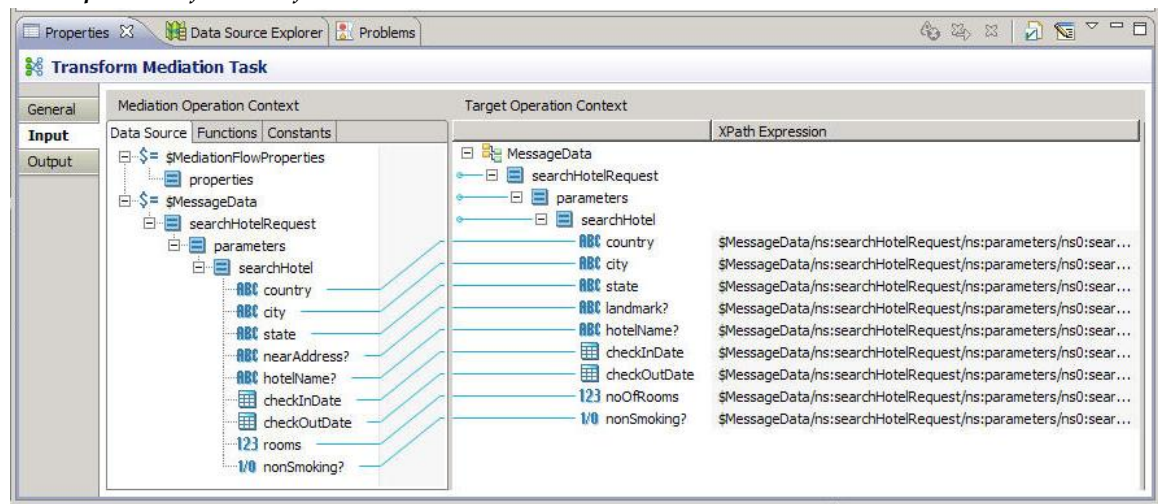
Transform tasks, used to manipulate data available in a mediation exchange, are necessary when the schema of the input, output, or fault message does not match the schema of the message of the expected recipient.

Transform tasks enable to achieve several goals:

- Create a mediation operation that allows new clients to use legacy services with different schemas. Your new client might need a service that returns an integer for salary information, but the legacy service returns a string.
- Contribute data to the mediation exchange for use in subsequent mediation tasks. For example, you might want to place into a string the time a message was sent, the sender of the message, and the value of one of the elements within a message. You can then use a log task to write the contents of that constructed string to the log file.
- Manipulate and store data in the mediation exchange without changing the actual message content.

Transform tasks have an **Input** tab that contains the expected schema of the recipient's message and the data available in the mediation flow.

The Input tab of a Transform task



The message panel contains an XSLT stylesheet that creates the message that the recipient expects. The message panel initially displays the expected schema of the recipient's message, to give you hints about constructing the message.

The Mediation Operation Context panel contains the data available from the message sender. You can drag items from the context panel to the message panel to perform simple mappings. More complex mappings are also possible through the XPath expression field and by using the right-click menu in the message panel to add XSLT statements.

XPath and XSLT are standard tools for data transformation. Extensive knowledge of XPath or XSLT is not necessary to use the mapper effectively. You can accomplish most transformation usage cases with information available in this chapter and in the Help text available for each XPath function in the product.

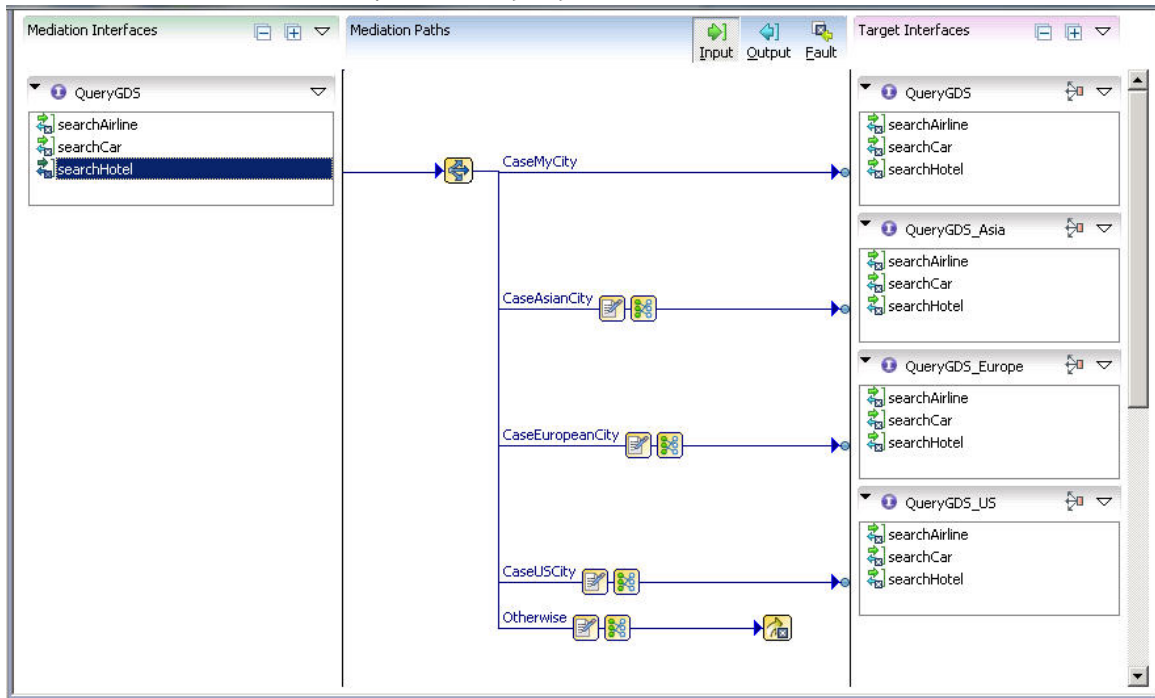
If you perform more complex transformations, however, it is helpful to have some detailed references on XPath and XSLT. It is beyond the scope of this manual to provide a complete reference for these tools. You can find the complete XPath and XSLT specification at www.w3.org, and several third-party commercial books are available on both XPath and XSLT.

Example of Transformation

In this example of using the Transform task, the mediation flow is for a travel reservation service.

The mediation operation exposes the service as a single interface, but the mediation flow routes incoming requests to the appropriate local service based on the location of the hotel. Different continents have different target services that perform the hotel reservation. The schemas for different locations are slightly different, and so some transformation might be necessary.

A travel reservation mediation flow: the input path



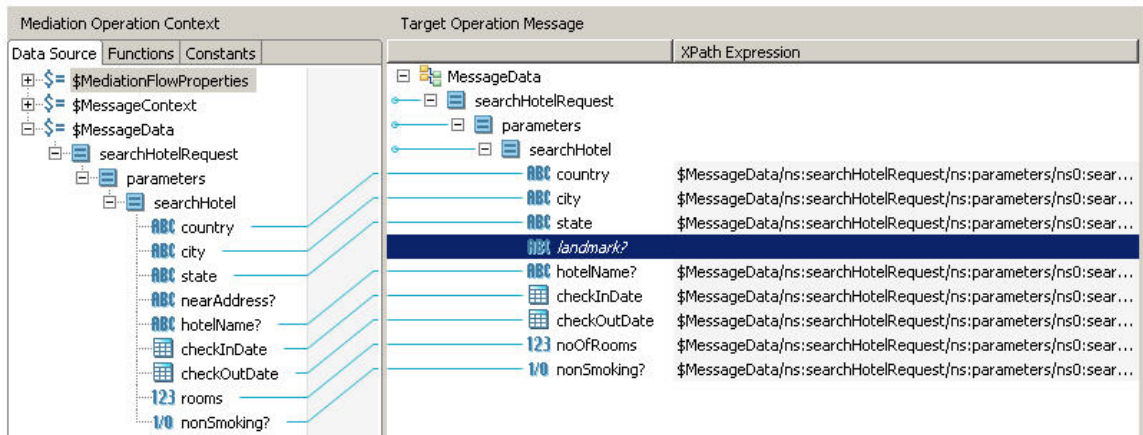
Transform tasks are required when requests come in for any city other than the local city, because the schemas for the other target operations are different from the mediation operation. The image shows the transform task for the case when a reservation is requested for an Asian city.

Basic Mapping

The schemas for the mediation operation and the target operation are similar, except that the mediation operation has an element named `nearAddress` and the corresponding element in the target operation is named `landmark`. For all other elements, you can drag and drop the data component in the mediation operation to the corresponding data component in the target operation, and the appropriate XPath expression is placed in the **XPath** field.

For the `nearAddress` and `landmark` elements, you might need to manipulate the data to transform it to the expected format.

A basic mapping example

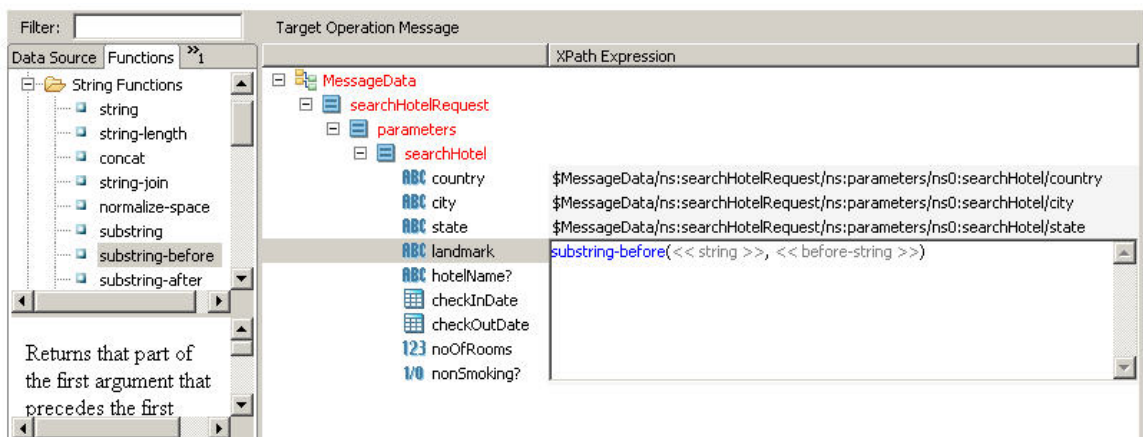


In this example, the `nearAddress` element contains the name of the location separated by a comma, followed by the actual address. The `landmark` element is expecting only the name of the location. To make the data match the target operation's expectations, you need to take the substring of the `nearAddress` element up to the comma that separates the name of the location from the address.

Using XPath Editor

When you drag functions from the **Functions** tab to the XPath Expression window, the function shows markers in double angle brackets (for example, `<<string>>`) for completing the function. You can drag data components and constants from the **Data Source** or **Constants** tab to complete the function. You can also type in the XPath Expression window to replace the markers manually.

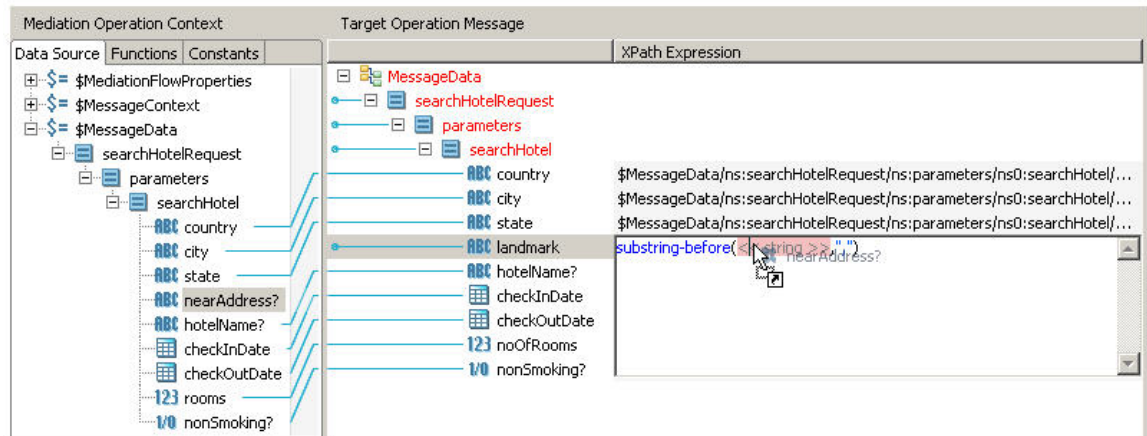
The XPath Editor Window



Procedure

1. Click the XPath Expression field next to the `landmark` element in the target operation message schema.
The field expands to a larger text box so that you can edit the expression easily.
2. Click the **Functions** tab on the at the top of the context panel.
3. Expand the String functions folder in the functions list and locate the `substring-before` function.
4. Drag the `substring-before` function into the XPath Expression window.
In the example, you would replace the `<<before-string>>` marker with a comma and then drag the `nearAddress` element onto the `<<string>>` marker. The image shows dragging the data components into an XPath function.

Dragging a data element into a function



More complex transformations are possible with the features available in the **Input** tab.

Data Contribution to the Mediation Exchange

The Transform task can modify the message data within the mediation exchange, and contribute new data to the mediation exchange.

On the **General** tab of the Transform task, the checkbox labeled **Contribute Output to Mediation Exchange** specifies how the Transform task results are handled.

- If **Contribute Output to Mediation Exchange** is cleared, the results of the transformation is used to construct a new message. This option is cleared by default.
- If **Contribute Output to Mediation Exchange** is selected, the results of the Transform task are added to the mediation exchange as a new data element. The new data element is available to subsequent mediation tasks along the same path, and the name of the data element is the same as the name assigned to the Transform task.

The **Contribute Output to Mediation Exchange** option is automatically selected if you use an external stylesheet for data transformation. See [External Stylesheets for Data Transformation](#).

External Stylesheets for Data Transformation

External, third-party XSLT stylesheet can be used for data transformation using the Transform task. This enables you to specify the transformation mapping in your workspace, outside the mediation flow.

It is possible to specify an external XSLT stylesheet for transformation in two ways using reference types.

- A static reference can be used to select a single (static) stylesheet from a folder in your project.
- A dynamic reference can be used to select a set of stylesheets from a folder in your project. At run-time one of the stylesheets in the list is used dynamically, based on the value provided for the *stylesheetURI* element in the **Input** tab of the mediation task.

For example, if the folder specified for the dynamic reference is *MySOAPProject/Service Descriptors* and the stylesheet is in the folder *MySOAPProject/Service Descriptors/folder1/sample.xsl*, the value that must be provided for the *stylesheetURI* element would be *folder1/sample.xsl*.

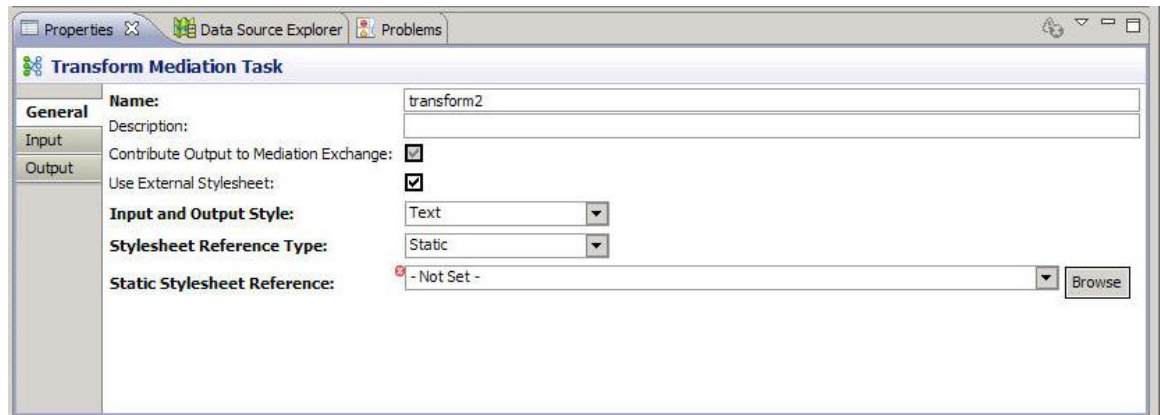


The stylesheet for a reference must be located in the same project as the mediation flow that uses it.

Specifying an External Stylesheet for Data Transformation

Procedure

1. On the **General** tab of the Transform task, select the checkbox labeled **Use External Stylesheet**. The stylesheet selection options open on the **General** tab.

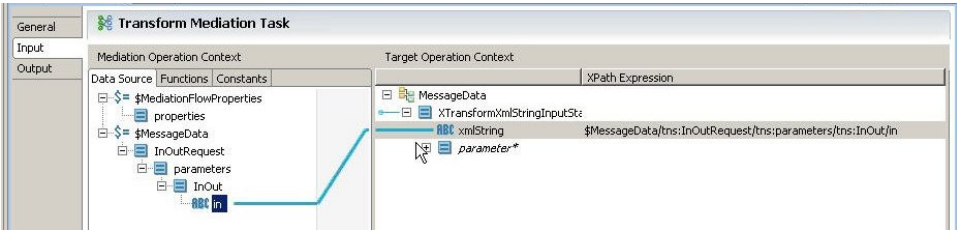


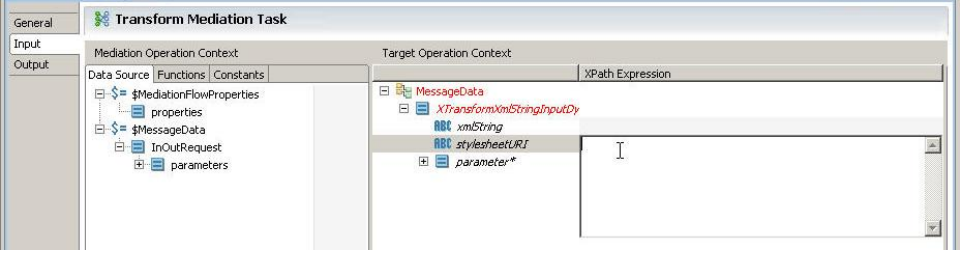
2. Open the Input Style drop-down menu and specify how the XML should appear.
 - **Text** Specified with a string.
 - **Binary** Specified with a binary value.
 - **Tree** Specified with a type of any, enabling you transform data that is already in an XML document



Contribute Output to Mediation Exchange is automatically selected for this type of transformation, which prevents the MessageData from being overwritten when an external transformation is used. Also, the input and output of the transformation task always match the Input Style you select. For example, if the input is text, the output is also be text.

3. Open the **Stylesheet Reference Type** drop-down menu of the **General** tab and select the type of reference for the Transform task to use:

Option	Description
Static reference type	<ol style="list-style-type: none"> 1. Browse and select a stylesheet from the stylesheets you have saved in your project. 2. Open the Input tab of the Transform task and map the data, so that when the data arrives, the value is transformed using the specified stylesheet. 

Option	Description
Dynamic reference type	<ol style="list-style-type: none"> 1. Open the Dynamic Stylesheet Folder drop-down menu of the General tab and select the folder where one or more stylesheets are located. 2. Open the Input tab of the Transform task and provide the stylesheet name as a parameter in the message data. 

At run-time, ActiveMatrix searches for this name in the folder you specified.

If the xsl file is in a sub-folder, the name must include the relative path name. For example, in the case where the xsl file is located in *company/dept/app.xsl*, the top-level folder (in this example, /toplevel) is prepended to locate the exact location for the file in the project:








/toplevel/company/dept/app.xsl



See [Transform](#) for reference information about the Transform task.

Schema Components

The message panel and context panel each contain schemas that contain data components. The icons represent the general data type of the component. To see the exact data type, hover the cursor over a component to open a pop-up.

Icons for schema components

Icon	Description
	Complex element that is a container for other datatypes. This is also called a parent in the schema tree.
	String or character value.
	Integer value.
	Decimal (floating point) value.
	Boolean value.
	Date value.
	Time value.

Icon	Description
	Binary (base 64()) value.
	Choice of multiple values. The actual data value can be one of a specified set of datatypes.



To improve performance, data contained within schema components in the left side of the mapper are not validated against the message schema for the operation. Therefore, data used within XPath expressions on the right side of the mapper are treated as untyped strings.

To perform datatype-dependent comparisons or operations, use the Constructor Functions on the Functions tab (for example, `xsd:int()`) to correctly specify the datatype. For example, to add two integers, the XPath expression would be:

```
xsd:int($MessageData/int1) + xsd:int($MessageData/int2)
```

Qualifier Characters

Schema data components can have additional characters to the right of the element name that specify additional information. If there is no qualifier, the schema component is required and you must provide a mapping that results in a value for the schema component.

Qualifier	Description
none	Element is required.
?	Element is optional.
*	Element repeats zero or more times.
+	Element repeats one or more times.

Context Panel

The name of the context panel is based on the type of path where the Transform task appears.

Type of Path	Name of Context Panel
Input	Mediation Operation Context
Output	Target Operation Context
Fault	Mediation Fault Context

The context panel always displays the schemas that define the data for the current mediation properties, message flow context, and message data. Regardless of the type of path, the schema of the mediation properties and message flow context are always the same. The schema for the message data varies depending upon the schema of the recipient's expected message.

Schema for message properties and message flow context

Schema Component	Description
MediationFlowProperties	This schema component contains an element named properties that is of type complex that contains the properties defined on the Properties tab of the mediation flow. See Adding a Mediation Flow Property .
MessageFlowContext	This schema component contains the defined context parameters. See Working with Message Context Properties .
MessageData	<p>The MessageData component contains the message of the expected recipient.</p> <p>For example, for input paths this component contains the schema of the input message of the mediation or the target operation. For output paths, this component contains the schema of the reply message of the mediation or the target operation. Similarly, for fault paths this component contains the schema of the fault message. For fault paths, this component contains a choice element that contains either one of the faults returned by the target operation or a generic Undeclared fault message</p>

Message Panel

The message panel contains the schema of the message that the recipient expects. The name of the message panel is based on the type of path where the Transform task appears.

Message panel

Type of Path	Name of Context Panel
Input	Target Operation Context
Output	Mediation Operation Context
Fault	Mediation Fault Context

You can use the data in the schemas from the context panel to construct the content of the message expected by the receiver. The message panel is actually an Extensible Stylesheet Language Transformation (XSLT) template that specifies how data will be transformed to produce the expected message.

You do not need detailed knowledge of XSLT to create the mappings for the message. Most mappings can be accomplished by simple dragging from the context panel to the message panel, and also possibly using a few XPath functions for simple data manipulation. If you want to see the XSLT template that is created from your mappings, click the **Show Edit Tab** icon on the toolbar, then click the **XSLT Source** tab at the top of the XPath editor dialog.

See [Data and Function Tabs](#) and [Managing Mappings](#) for more information about using XPath functions and creating mappings.

Data and Function Tabs

Use the tabs at the top of the context panel to select items to drag to the message panel.

Data and Function Tabs

Tab	Description
Data Source	Contains the schemas for the mediation flow properties, message flow context, and message data. This tab is selected by default when you view the Input tab. See Context Panel .
Functions	<p>Contains a set of XPath functions organized into related functional groups. XPath (XML Path Language) is an expression language developed by the World Wide Web Consortium (W3C). XPath functions perform data manipulation, such as mathematical functions, string manipulation, or logic operators.</p> <p>You can select and drag XPath functions in this tab to the XPath expression field or to the Show Edit Tab dialog in the message panel.</p> <p>Each function has help text to describe the function's use and syntax. The help for the function is displayed below the function list in the Functions tab.</p> <p>See TIBCO XPath Functions that describes the functions added by Mediation.</p>
Constants	Contains constants such as whitespace or symbol characters that can be used in XPath expressions.

When you drag data, a function, or a constant to the right-hand panel and hover over an existing expression in an XPath editing window, the background color of the text underneath the cursor changes. The new color indicates the result of placing the item at that point:

- Light turquoise - The highlighted text is the first parameter of the dropped function.
- Light pink - The dropped item replaces the existing text.



As noted in [Schema Components](#), data in schema components on the **Data Source** tab are not validated and checked against the types in the message schema. Therefore, data is coerced into an untyped string. The Constructor Functions on the **Functions** tab must be used on data to correctly evaluate most functions and operators.

TIBCO XPath Functions

TIBCO XPath Functions describe the functions added by Mediation.

TIBCO XPath Functions

Function Name	Description
base64ToString	<p>Converts a base64 binary encoded string to a string using the specified encoding. If encoding is not specified, UTF-8 is used.</p> <p>Template</p> <pre>base64ToString(<< encodedString >>, << optional encoding >>)</pre> <p>Return Type</p> <p>string</p>
stringToBase64	<p>Converts a string to a base64 binary encoded string.</p> <p>Template</p> <pre>stringToBase64(<< stringToEncode >>)</pre> <p>Return Type</p> <p>string</p>
hexToString	<p>Converts a hex string to a string using the specified encoding. If encoding is not specified, UTF-8 is used.</p> <p>Template</p> <pre>hexToString(<< encodedString >>, << optional encoding >>)</pre> <p>Return Type</p> <p>string</p>
stringToHex	<p>Converts a string to a hex encoded string.</p> <p>Template</p> <pre>stringToHex(<< stringToEncode >>)</pre> <p>Return Type</p> <p>string</p>
timestamp	<p>Returns the number of milliseconds since midnight, January 1, 1970 UTC, at the instance of the call to this function</p> <p>Template</p> <pre>timestamp()</pre> <p>Return Type</p> <p>long</p>

Creating Custom XPath Functions

Procedure

1. Run TIBCO Business Studio from the Start menu.
For example, select **Start > All Programs > TIBCO_Home > TIBCO Business Studio N.N > Studio for Designers**
2. Select **File > New > Project**
3. In the New Project dialog under **Plug-in Development**, select **Plug-in Project** and click **Next**.
4. Specify a name for the project that reflects the XPath functions (for example, My Custom Functions). Accept all other defaults and click **Next**.
5. On the Plug-in Content page, locate the Plug-in Options group, and clear these options:
 - Generate an activator, a Java class that controls the life cycle of the plug-in.
 - This plug-in will make contributions to the UI.
6. Accept all other defaults, and click **Next**.
7. In the Templates page, select **Custom XPath Function Wizard** and click **Next**.
8. In the **Custom XPath Function Group** dialog box, provide values:
 - Category: This is the name of the category that will include the custom XPath function.
 - Prefix: The prefix for the functions
 - Namespace: The namespace for the functions.
 - Help Text: The description of the functions.
9. Click **Next** to continue.
The XPath Function Group Creation Section dialog is displayed. Here you specify the function and function parameters.
10. Click the **Add** button located on the right side of the XPath Function table and provide values.
 - Name: The name of the function.
 - Return Type: The return type of the function.
 - Description: The description of the function.
11. Click the **Add** button located on the right side of the XPath Function Parameters table.
 - Name: The name of the parameter.
 - Type: The data type of the parameter.
 - Optional: Select the check box if the parameter is optional.
12. Click **Finish**.
13. TIBCO Business Studio opens the Open Associated Perspective dialog, which asks if you want to open the Plug-in Development perspective.
 - Optionally, select the check box **Remember my decision**. Click **Yes**.
TIBCO Business Studio opens the custom XPath function plug-in and the Plug-in Development perspective.

Result

Along with the custom XPath plug-in, a SOA Project *<plug-in project name>.deploy.soa* is created.

Your custom code is written in `<plug-in project name>\src\<plug-in project name>\<category name>.java`.

Exporting Custom XPath Functions

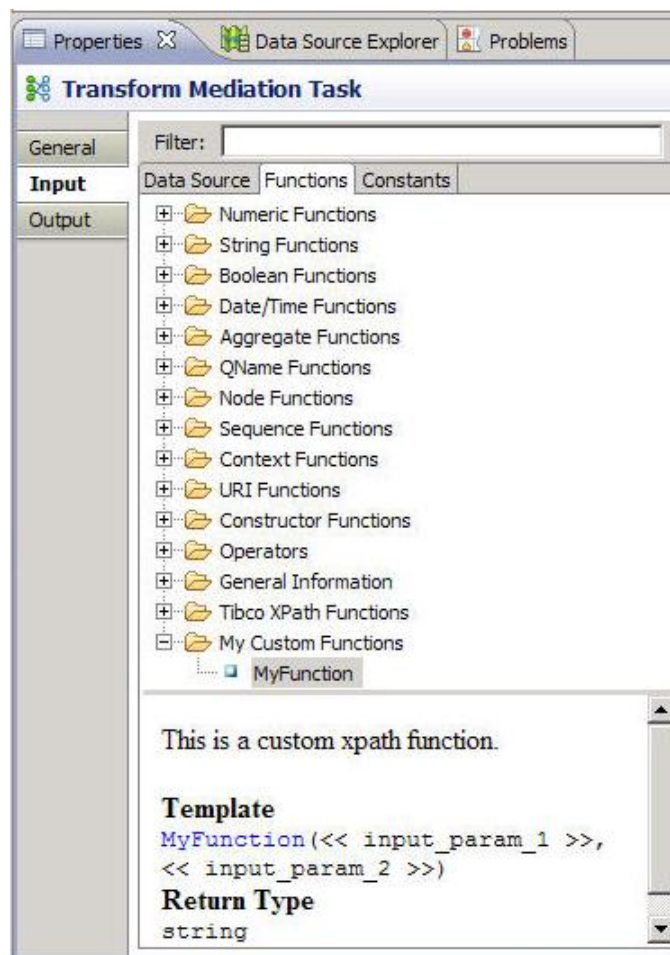
You can install a custom XPath function in TIBCO Business Studio.

Procedure

1. To create a feature project specify the plug-in to package into the new feature.
See Supplemental **Eclipse Help > Plug-in Development Environment Guide > Reference > Wizards and Dialogs > New Project Creation Wizards** for more information.
2. Export the feature project
Make sure you select the check box for the **Generate metadata repository** option.
See Supplemental **Eclipse Help > Plug-in Development Environment Guide > Reference > Wizards and Dialogs > Export Wizards > Export Feature** for more information.
3. Install the feature using **Help > Install New Software**.
Specify the location where you exported the feature project. Unselect the check box for the Group items by category option which will then list the feature project.

Result

The custom XPath function is ready for use and can be accessed from the Input path of the data transform function.



Deploying Custom XPath Functions

After the file `< plug-in project name>\src\<plug-in project name>\<category name>.java` is updated with the custom code, the deployable artifacts can be generated.

Procedure

1. Make sure the Target Platform points to TIBCO ActiveMatrix Runtime and no errors occur in the custom XPath function plug-ins.
See *Composite Development* for information on switching the Target Platform.
2. In the Project Explorer pane, expand the `< plug-in project name>.deploy.soa project`.
3. Expand the Composites folder.
4. Right-click `< plug-in project name>.apt.composite` and click **Create DAA**.

Result

The **Create Deployment Archive** wizard is invoked. Refer to *Composite Development* for more information on using this wizard.



Deploy this deployment archive, the DAA, like any other SOA project. Refer to *Administration* for information on uploading and deploying the DAA.

Testing Custom XPath Functions

Custom XPath function can be tested in RAD by creating a Run As or Debug As configuration

Procedure

1. Add one of the following to the Functions list along with the main composite:
 - A composite generated by the Custom XPath Function wizards to the list.
 - A DAA created from the composite.



Make sure that the composite or DAA that holds Custom XPath Function is at the top of the list of Composite/DAA(s), before the SOA DAA/Composite.





2. Select **Apply** and **Run/Debug**.

Mapper Toolbar Buttons

The toolbar contains icons to perform various functions in the mapper.

Mapper toolbar buttons

Button	Description
	Pins the property view to the current selection.
	Click this button to view errors for the selected element or children.

Button	Description
	Click to remove the selected mapping. This button is available only when a mapping is selected in the message panel. If you remove the mapping when a parent node in the schema tree is selected, all mappings for child nodes of the parent are also removed.
	Deletes XSLT statements that you have added using the right-click menu, such as variables, comments, or choose statements. This button is available only when a statement you have added is selected.
	Opens the Show Check and Repairs dialog. See Repairing Incorrect Mappings .
	Opens a larger Show Edit Tab XPath editing window for the selected element in the message panel. The window gives you access to a larger XPath viewer, the XSLT source, and controls that enable you to further edit the XSLT statements. Click this icon a second time to make the Show Edit Tab XPath editing window disappear.

Right-Click Menu in the Message Panel

Right-clicking on a data component in the message panel opens a popup menu with several choices.

Right-click menu in the message panel

Menu Item	Description
Show Mappings	Expands the selected component to show all sub-components with mappings. Also expands any data components in the left-hand panel that correspond to mappings so that all mapping lines are shown. If no component is selected, the operation is performed on the root of the schema tree.
Show Errors	Expands the selected data component to show all sub-components that have errors. If no component is selected, the operation is performed on the root of the schema tree.
Expand All	Expands all sub-components of the selected data component. If no component is selected, the operation is performed on the root of the schema tree.
Surround With > Surround With Choose	Surrounds the selected data component with a Choose statement. See Surrounding a Component With a Choose Statement .
Surround With > Surround With If	Surrounds the selected data component with an If statement. See If Statements .

Menu Item	Description
Surround With > Surround With ForEach	Surrounds the selected data component with a For Each statement. See For Each Statements .
Surround With > Surround With ForEach Group	
Add Child > Variable	Adds a sub-component to the selected data component. The child component will be a variable. Variables can be set to a constant value and used in other mappings in the message panel. See Adding a Variable to a Mapping .
Add Child > Comment	Adds a sub-component to the selected data component. The child component will be a comment. See Surrounding a Component With a Choose Statement .
Add Sibling > Variable	Adds a data component at the same level as the selected component. The new component will be a variable. Variables can be set to a constant value and used in other mappings in the message panel. See Adding a Variable to a Mapping .
Add Child > Comment	Adds a data component to the same level as the selected component. The new component will be a comment. See Surrounding a Component With a Choose Statement .
Toolbar icons	The selections from the toolbar are also available in the right-click menu. See Mapper Toolbar Buttons .

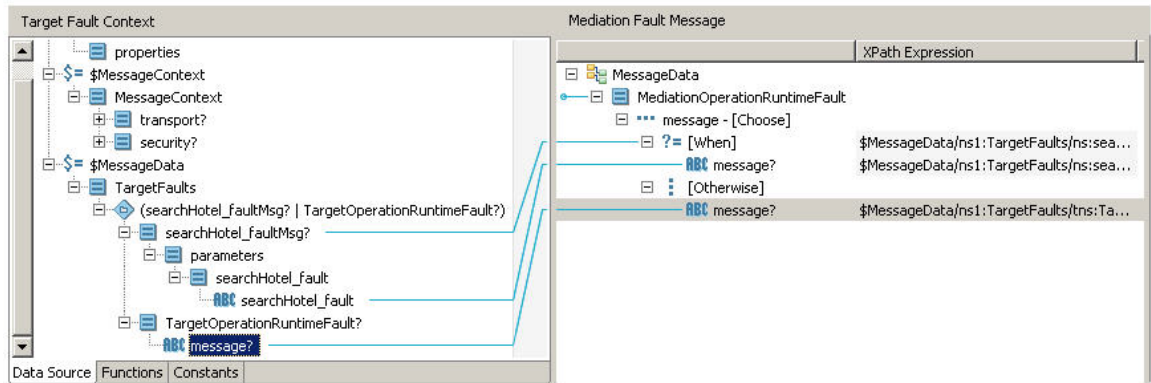
Surrounding a Component With a Choose Statement

Choose statements enable you to conditionally specify the mapping based on an expression. Choose statements consist of a When clause to specify the condition you want to test, the mapping you want to perform if the condition is true, and an Otherwise clause to contain a mapping to perform if no conditions evaluate to true.

Procedure

1. Select the component to surround, right-click, and choose **Surround > Surround with Choose...** from the menu.
 2. In the Surround With Choose dialog, enter the number of When conditions to test against, and also specify whether to include an Otherwise clause for any unhandled conditions.
 3. For each When clause, create an XPath expression that evaluates to a boolean.
 4. Under each When clause, provide the XPath expression for the mapping that occurs if the When condition evaluates to true.
 5. If an Otherwise clause is specified, provide an XPath expression for the mapping that occurs if no When conditions evaluate to true.
- An example of using a Choose statement is when more than one fault message is handled by the same Catch Fault task. The figure below shows a Transform task on a fault path that handles two faults. The Choose statement specifies that when the searchHotel_faultMsg is returned, send the value of the searchHotel_fault element. Otherwise, send the value of the message element.

An example of choose statement

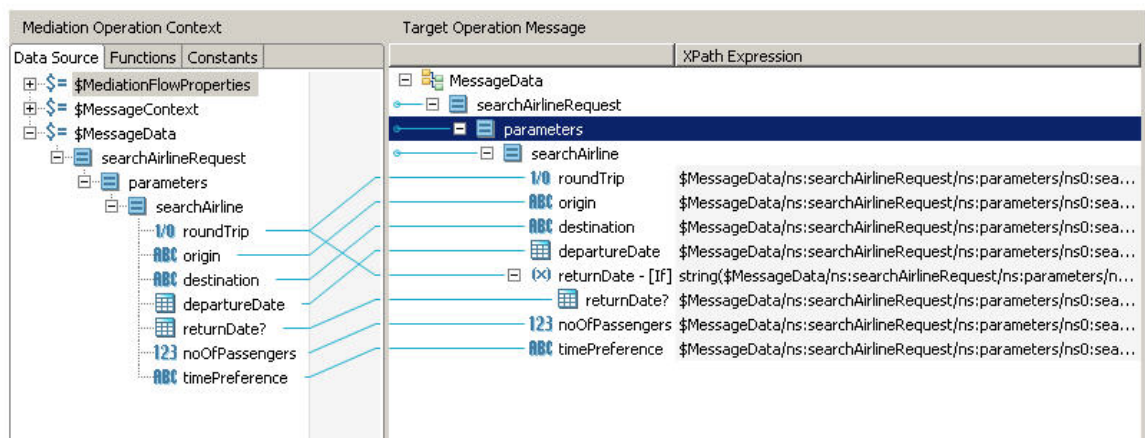


If Statements

If statements enable you to specify a condition, and if the condition is met, then the specified mapping is output.

When you chose this option, an If statement appears before the selected element, and you must place an XPath expression in the If statement that evaluates to a boolean. If the expression evaluates to true, the specified mapping is performed. If the expression evaluates to false, the mapping is not performed and no value is set for the item. Do not place an If statement around schema data components that are marked as required.

Example of If statement



The example requires a comparison of the value of a boolean element. To obtain the value of the element, the element is coerced into a string using the `string()` function and then compared to the value of the string "true". See [Testing Boolean Values](#) for more information about comparing the value of boolean data components.

In this example, the `returnDate` schema element is optional. The `returnDate` element is surrounded by an If statement that evaluates whether the `roundTrip` element is true. If `roundTrip` is true, then the element is output, if `roundTrip` is false, the `returnDate` element is not output. The expression in the If statement is:

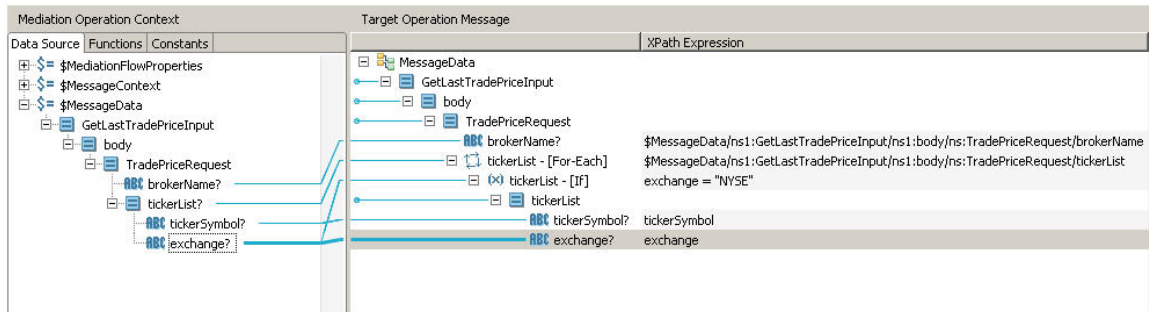
```
string($MessageData/namespace('ns')/searchAirlineRequest/namespace('ns')/parameters/namespace('ns0')/searchAirline/roundTrip) = "true"
```

For Each Statements

For Each statements enable you to execute one or more statements once for each data element in a list.

When you choose this option, a For Each statement appears before the selected data component, and you must place an XPath expression in the For Each statement that evaluates to a list of zero or more items. This is useful when you want to manipulate sequences or repeating elements.

Example of For Each statement



In this example, the requestor sends a list of ticker symbols and the stock exchanges on which they are traded. The mediation flow routes the request to different services for each stock exchange. The For Each statement takes the list of ticker symbols and executes the remaining statements once for each symbol in the list. The If statement examines the exchange element and outputs only the ticker symbols for the "NYSE" stock exchange.

Adding a Variable to a Mapping

Variables can be used in any XPath expression within the message panel.

Choosing this option opens a dialog that enables you to specify the name of the variable. You can change the name of the variable at a later time by selecting the variable and clicking the **Show Edit Tab** button in the toolbar. The **Variable Name** field can be used to change the variable's name.

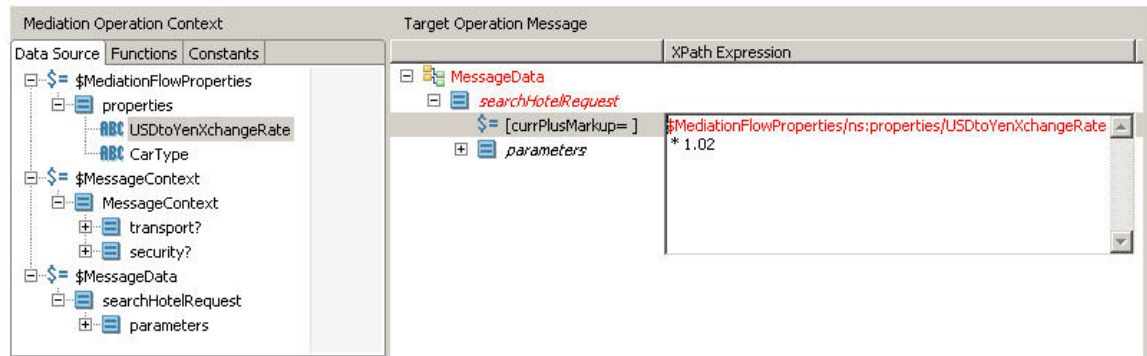
Procedure

1. The value of the variable is specified by supplying an XPath expression, either by mapping data from the context panel or by using XPath functions or constants.
2. Once the variable's contents have been supplied, the variable can be referenced within the scope that it has been defined. That is, you can reference a variable from within the same component or within sub-components of the component in which the variable is defined.

Adding a variable is useful if you perform the same computation repeatedly. You can refer to the results of the computation in several message elements instead of recreating the computation for each item.

In this example, the variable uses the mediation flow property *USDtoYenXChangeRate* to get the value of the current exchange rate. That value is then multiplied by 1.02 to add a 2% markup. The variable can then be referenced in subsequent statements in the mapping.

An example of adding a variable to a mapping



Managing Mappings

A mapping correlates data from the schema in the context panel with a data component in the message panel.

Procedure

- You can create and manage mappings using several functions:

Function	Procedure
Creating a mapping	<ol style="list-style-type: none"> Drag and drop data components from the left-hand panel to the right-hand panel. The appropriate XPath expression is displayed in the XPath Expression field.
Adding functions or constants to an XPath expression	Use the tabs at the bottom of the context panel (described in Data Function Tabs).
Opening a larger window in which to view or edit an XPath statement	Click an expression in the XPath Expression field, or click the Show Edit Tab button on the toolbar.

- Lines appear between data components that are mapped to each other. The lines are blue when both components are visible, but the lines turn into a dashed green line when one or more mapped components are collapsed into its parent in the schema tree.
- Data components in the message panel are initially displayed in italics. Italic text indicates that the components are hints to the potential mappings you can create.
- Once you create a mapping for a data component, the hint changes from italics to non-italic font. Non-italic font indicates that the mapping is now an XSLT statement that transforms the data into the specified component. You can change a hint into a statement without performing a mapping by selecting a component in a message schema and dragging it past the dividing line between the left and right panels.
- Data components on the right-hand side of a mapping can be either black or red. If the component is black, the XSLT statement for the component is valid and complete. If the component is red, that indicates the statement for the component is an error and must be repaired as described in the next section.

Repairing Incorrect Mappings

Any incorrect statements are displayed in red in the message panel. Errors can occur for a number of reasons. Correct any errors before attempting to execute your mediation flow. The reasons for errors could be:

- A required component has no statement and therefore must be specified.
- The message schema has changed, and existing statements may no longer be valid.
- The XPath formula for a component may contain an error.

Procedure

1. If you hover the cursor over any red component name in the message panel, a pop up describing the error opens.
2. Find potential problems in your mappings and correct them.
 - a) Click the **Show Check and Repairs** button on the toolbar. This button opens a dialog with all potential problems in the specified mappings.
 - b) Select the **Fix** checkbox for potential errors, and the software will attempt to automatically fix the problem.
Some potential problems in the Show Check and Repairs dialog cannot be fixed easily, and no check box for these items appears in the **Fix** column. For example, if a component expects a string and you supply a complex type, the corrective action to fix the problem is not clear. The problem cannot be automatically fixed. You must repair these items manually.
3. You can delete mappings by selecting one and clicking **Remove Mappings**.
 - If a child component is selected, the component is returned to its original state and no mapping is specified.
 - If a parent component is selected, mappings for all child components are also removed.

Mapping an Empty Complex Type

Procedure

1. In the Generate Reply Mediation Task pan select an object.
2. Drag the empty complex object from the **Target Operation Context** pane on the right to the **Mediation Operation Context** pane on the left side.

Using XPath

The Input tab uses XPath as the language for locating and manipulating data. XPath (XML Path Language) is an expression language developed by the World Wide Web Consortium (W3C) for addressing parts of XML documents.

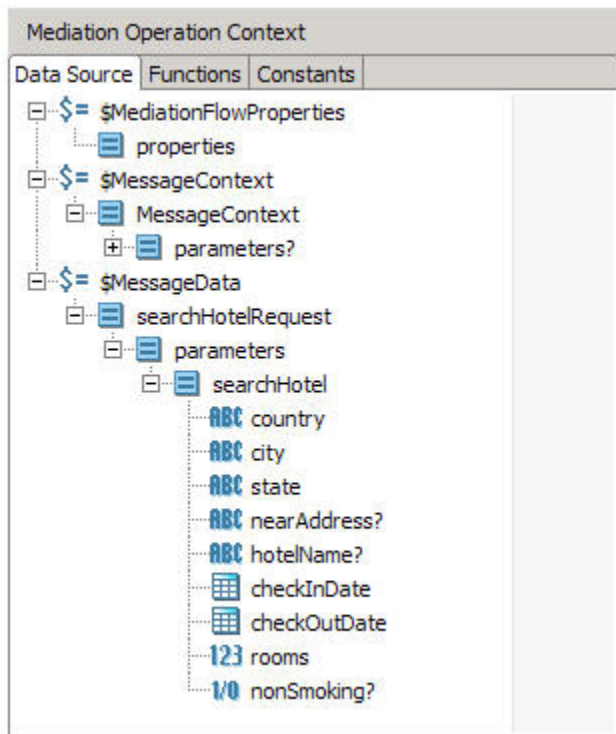
XPath also provides basic manipulation functions for strings, numbers, and booleans. To use XPath in the Input tab, you need only be familiar with the basic XPath concepts, but you might want to learn more about XPath when building complex expressions. For a complete description of XPath, refer to the XPath specification (which can be obtained from www.w3.org).

Addressing Schema Components

All data in the context and message panel is represented as an XML schema. The data can be a simple (strings, numbers, booleans, and so on) or a complex component. Complex components are structures that contain other schema components, either simple components or other complex components. Both

simple and complex components can also repeat. That is, they can be lists that store more than one component of the given type.

XPath is used to specify which schema component you would like to refer to. For example, this schema may be available in the context panel.



The context panel of the example shows the schema available for a mediation operation.

Three top-level items, each a root node in the context panel, are present:

- MediationFlowProperties
- MessageContext
- and MessageData

Each of these nodes has its own associated structure. MediationFlowProperties has a complex component named properties and MessageData has a complex component named searchHotelRequest.

References to a particular data item in any of these schema start with the root node and slashes (/) indicate a path to the data component. For example, the country element in the SearchHotel complex component that is in the parameters component would look like this in an XPath mapping field:

```
$MessageData/searchHotelRequest/parameters/searchHotel/country
```

The path starts with a dollar sign, then continues with node names using slashes, like a file or directory structure, until the location is named.

Some schema components must be prefixed with their namespace prefix. The prefix is automatically added to components that require this when dragging and dropping data in the XPath Expression field.

Evaluation Context

XPath also has a method for referencing relative paths from a particular node. If you have an evaluation context, or a particular starting node in a schema tree, you can specify the relative path to other elements in the tree.

For example, if your evaluation context is `$MessageData/searchHotelRequest/parameters/searchHotel`, you can reference the sub-items of `ShipName` without specifying the entire path. If you want to reference `$MessageData/searchHotelRequest/parameters/searchHotel/country`, the relative path would be `./country`. The path is relative to the evaluation context — `country` is at the same level in the schema tree as the evaluation context.

Search Predicates

An XPath expression can have a search predicate. The search predicate is used to locate a specific element of a repeating schema item. For example, consider a schema where the `$MessageData/searchReservations/todaysReservations` item is a repeating element. If you want to select only the first item in the repeating element, you would specify this:

```
$MessageData/searchReservations/todaysReservations[1]
```

The `[1]` specifies the first element of a repeating item.

Sub-items can also be examined and used in a search predicate. For example, to select the element whose `reservationId` is equal to `"3A54"`, you would specify:

```
$MessageData/searchReservations/todaysReservations[reservationId= "3A54"]
```

In the example above, the evaluation context of a predicate is set to the item containing the predicate. Therefore, `reservationId` is assumed to be within the `todayReservations` component.

You can also use functions and expressions in the search predicate. For example, if you want to find all elements after the first, you would specify:

```
$MessageData/searchReservations/todaysReservations[position()>1]
```

Testing Boolean Values

To test the value of a boolean node, you can use the `data()` function to obtain the value of the node. A common error in XPath functions is to supply a boolean node in a condition and expect that the condition will evaluate to true or false based on the value in the node. For example:

```
if ($MessageData/searchHotelRequest/parameters/searchHotel/nonSmoking) then ...
```

The condition in the `if` statement above would return true when the `nonSmoking` component is present, regardless of whether the value of the component is true or false. To evaluate the value of a boolean element, use this expression:

```
if (data($MessageData/searchHotelRequest/parameters/searchHotel/nonSmoking))
then ...
```

You can also use the `string()` function to coerce the comparison to the string value of the Boolean node and then compare to the value of `"true"` or `"false"`. For example:

```
string($MessageData/searchHotelRequest/parameters/searchHotel/ nonSmoking) = "true"
```

Comments

You can add comments to XPath expressions using the XPath 2.0 syntax for comments. The syntax is:

```
{-- <comment here> --}
```

For example, this XPath expression contains a comment:

```
$MessageData/searchHotelRequest/parameters/searchHotel/country {-- returns the
country --}
```

Transforming XML with Related Tasks

In addition to the Transform mediation task, ActiveMatrix provides tasks that enable you to manipulate XML data in text, binary, or tree formats.

The Parse XML Task

The Parse XML task is used when you have an XML document stored in a string or binary field. This task produces a tree representation of the XML that can be used by subsequent tasks in the mediation flow. You can pair the Parse XML task with the Render XML task to convert the parsed XML back into a string or binary field for transmission within a message. See [Parse XML](#) for reference information about this task.

The Render XML Task

The Render XML task takes an XML tree for a specified schema and converts it to a string or binary element that contains the XML document. You can pair the Render XML task with the Parse XML task to convert the parsed XML back into a string or binary field for transmission within a message. See [Render XML](#) on page 195 for reference information about this task.

The Validate XML Task

You can use the Validate XML task to validate message data, a WSDL message, XML text, binary, or XML tree formats against a schema. The output of the Validate XML task is contributed to the mediation exchange, and can be used by downstream tasks. See [Validate XML](#) for reference information about this task.

In addition to the Validate XML task, the message received by the mediation component can be validated using the `VALIDATE_MESSAGE_DATA` property that is added by default to mediation flows. See [Validation of Message](#).

Querying a Database

The Query Database task performs a SQL SELECT statement on a database.

The task can specify three types of records:

- One or more tables in the FROM clause of the SELECT statement
- One or more columns to return in the SELECT list
- One or more conditions in the WHERE

You also have the option to specify the maximum number of rows to return.

The Query Database task can be used to look up data in a database table to enrich the data available in a mediation flow.

These are two usage scenarios:

- Store service names and namespaces for dynamically bound service references in a database table. You can then update the database table when a new service becomes available, and the mediation flow does not need to be changed to obtain the information about the new service.
- Use a database query to add information to an incoming request. For example, an incoming request may specify a US postal zip code, and a database query can be used to look up the city and state to add this information to the request.

JDBC Resource Templates

A JDBC resource template can be used to establish a connection to a database and obtain table and column information to complete the SELECT statement. This resource template is only used during design. When the mediation flow is used in a composite and deployed, the resource template is ignored.

Resource templates are defined on the mediation flow and are used to specify a resource (such as a database connection) that can be used by one or more tasks in a mediation flow.

The property specified on the Properties tab of the mediation flow provides the database connection used for each Query Database task.

To connect to more than one database or use different user accounts, create one resource template for each database connection. Query Database tasks that use the same resource template will use the same database connection.




Resource templates must be associated with JDBC resource templates at the component or composite level, and you can also override JDBC resources at deployment time in the ActiveMatrix Administrator interface. See *Composite Development Guide* for more information about resource templates and the Administrator interface.

Defining a Resource Template

Resource templates are defined on the mediation flow and are used to specify a resource (such as a database connection) that can be used by one or more tasks in a mediation flow.

Procedure

1. Navigate to the **Properties** tab of a mediation flow by clicking on the canvas of a mediation flow in the editor window.
2. Click the  icon to add a new property.

By default, the property name is specified as `propertyn` (where each newly added profile increments `n`). Specify a new name for the profile, if desired.

The value in the **Type** column must be JDBC Resource Template. This value is read-only.

3. In the **Value** column, click the ellipsis (...) and choose a previously defined template from the **Select JDBC Resource Template** dialog box.
4. Click OK.

Result

The JDBC Resource Template is created and is ready for use by the Query Database task.

Configuring a JDBC Driver

The JDBC driver referenced by the JDBC Resource Template must be configured before it can be used.

Prerequisites

Configure the JDBC Resource Template.

Procedure

- Navigate to **Window > Preferences > Data Management > Connectivity > Driver Definitions**.
 - You can optionally specify a JDBC Resource Templates for use while creating Query Database tasks. JDBC Resource Templates define connections to databases. See *Composite Development Guide*.

Registering a JDBC Driver

To connect to a database at design time from within the Query Database task, you must first register the JDBC driver.

Procedure

1. Navigate to the **Data Source Explorer** tab.
2. Right-click Database Connections and select **New...** .
The Connection Profile pane displays.
3. Select the driver from the list of Connection Profile Types and click **Next**.
The Specify and Driver and Connection Details pane displays.
4. Click the **New Driver Definition** icon location to the right of the Drivers drop-down list.
The New Driver Definition dialog box displays.
5. Select the JDBC Driver in the **Available driver templates** list of the **Name/Type** tab.
6. Navigate to the **Jar List** tab.
The jar file for the selected database is listed in the list of driver files..
7. Select the JAR file (generated according to the selected JDBC driver) and click **Remove JAR/Zip**.
8. Click **Add JAR/Zip**.
9. In the **Select the file** dialog box select the driver appropriate for your database, and click **Open**.
10. Click OK.

Result

The database is now registered and is ready to be used within the Query Database task.

Configuration Tabs of the Query Database Task


The configuration tabs of the Query Database task are described.

General Tab

- On the **General properties** tab, you can specify a name and description. You must also select one of the resource templates defined for the mediation flow. See [JDBC Resource Templates](#).
- The **Max Row Count** field specifies the maximum number of rows to accept from the query results. For example, a positive integer of 1 returns only one row. The choice of **Unlimited** allows an unlimited number of rows in the result set.
- The **Query Timeout** field specifies the timeout, in seconds, for a query statement to execute before an exception is thrown.

Query Tab

You use the Query tab to define the SELECT statement for the query.

If you specified a JDBC property in the mediation flow Properties tab, clicking the connection icon  opens a connection and compares the table and column data with the metadata from the database. If the connection is not successful, an error notifies you of the reason.

Three lists enable you to select tables, input data, and output columns for use in the WHERE clause of your SELECT statement.

Input data is used in the WHERE clause of your SELECT statement. Use the add (+) and delete (x) icons to the right of each list to add and delete items from each list:

- When a database connection is present and valid, the + icons display information from the database for selecting tables and output columns.
- When no database connection is present, the + icons allow you to add items to each list, but you must name each item and specify a type if necessary.

Clicking the + and x icons on the Input table attempt an automatic update of the WHERE condition. If you have modified the WHERE condition, the delete might not update it and you must fix it manually.

Use the Where Condition field on the Query tab to edit the WHERE clause of the query. You can add an input variable to a condition by typing a question mark (?) in the condition. Each input variable appears in the mapper panel on the Input tab, and you can supply data from the mediation exchange for the input variable. For example, if you want to create a condition where you look up a ZIP code supplied in the input message, you can add the condition `table.ZIP = ?`. When you add a question mark into the WHERE clause, an input variable appears in the Input Data list. Supply a name for the input variable, then data from the mediation exchange can be mapped to the input variable.

Table join conditions are never automatically added to the WHERE clause. To specify any join conditions for your query, you must manually edit the WHERE clause.

The **SQL Statement** field displays a read-only version of the query you have specified. Length parameters are stripped from the SQP Type, and only the base type is used in the mapping. For example, `char(12)` becomes `char`.

Supported SQL types and their mappings to XML

SQL/92 Data Type	XML Type Equivalent
TINYINT	short

SQL/92 Data Type	XML Type Equivalent
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	float
DOUBLE	double
CHAR	string
VARCHAR	string
NCHAR	string (multi-byte)
NVARCHAR2	string (multi-byte)
DATE	date
TIME	time
TIMESTAMP	dateTime

Vendor-specific types are cast to string. You can enable the mapper to automatically recognize these types in two ways:

- Force vendor-specific types to a compatible XML type using the mapper cast.
- Override the type that is retrieved from the database for the column to a similar SQL/92 type.



Binary or other complex data types such as JAVA_OBJECT are not supported.

Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transforming Data in a Mediation Exchange](#) for information on using a mapping panel.

Output Tab

The **Output** tab is a read-only display of the output schema for this task. The output schema is determined by the output columns selected on the **Query** tab.

Test Tab

Use the **Test** tab to supply test data for values of input variables and test the query against the database associated with the specified JDBC resource template. To test the statement, a valid database connection must be present.

You can use a custom JDBC driver to test the database query. For information about configuring a custom JDBC driver, see [Composite Development](#).

You must have a valid JDBC resource template associated with the shared resource profile used by this task. The JDBC shared resource is used only in the design environment.



Ensure that the JDBC resource template you use for testing in the design environment connects to a database that is similar to the database used when the project is put into production.

Dynamic Requests

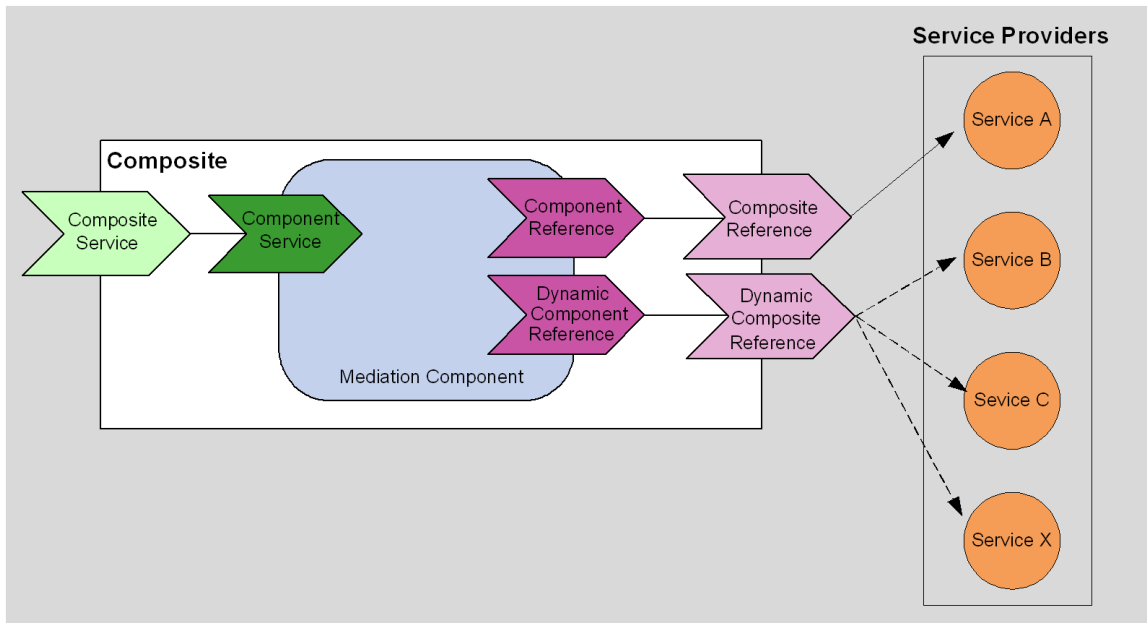
Dynamic binding enables routing of incoming requests to target services as they are needed at runtime.

The target interfaces in a mediation flow correspond to component references in an ActiveMatrix composite. Typically, a component reference is wired to a composite reference that points to a service provider. This static binding is specified when the mediation component and composite are designed, and the service binding is hard-coded into the composite.

Dynamic binding allows components to supply a reference to the service provider when the deployable application archive (DAA) created from the composite is running.

The below diagram shows the differences between static and dynamic binding in composite references.

Static and dynamic binding



The composite reference is statically bound to Service A. However, the dynamic composite reference can invoke Service B, Service C, or Service X without having to specify a static configuration at design time.

Dynamic references allow the component to specify which service to invoke. Therefore, new services can be started and a component can invoke those services without redesigning the composite and restarting the DAA created from the composite.

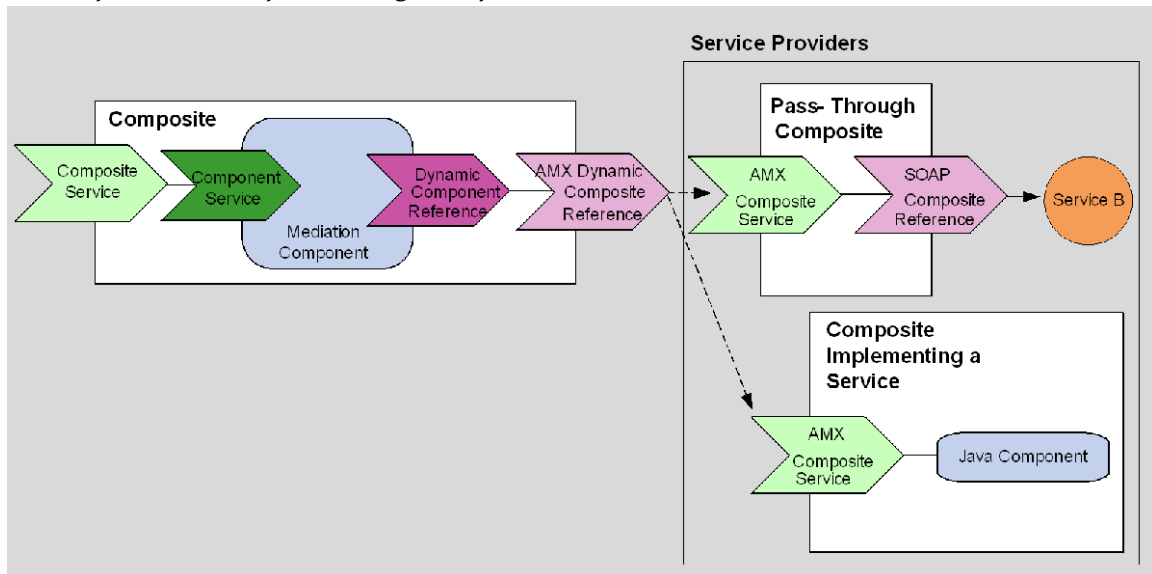
One example of using dynamic references is a set of services that return information for United States postal ZIP codes. The consumer sends a message to a mediation component containing the ZIP code. The service provider can implement a number of services for particular ZIP codes. When new ZIP codes are introduced, dynamic binding allows the service provider to create and start a new service for the new ZIP codes without changing any existing composites. Requests for information about new ZIP codes are handled without system downtime.

Service Providers for Dynamic Composite References

Dynamic composite references can refer only to bindings of type virtualization. That is, the service type in the provider composite cannot be JMS or SOAP. If your service provider uses the SOAP or JMS protocol, you can create a simple pass-through composite that passes the message to the ultimate service provider.

The diagram shows a dynamic composite reference using a composite that implements a service and also using a pass-through composite for referencing a SOAP service.

Service providers and pass-through composites



Referring to Service Providers

The component implementation determines the service that is invoked for a dynamic reference. To specify the service, the implementation supplies the application name and service name. ActiveMatrix resolves the application name and service name to the correct running service.

The service name is the name specified for the promoted service in the composite.

Configuring Dynamic Binding

There are four high-level steps for configuring dynamic binding. Perform the four procedures to configure an application to use dynamic binding.

Procedure

1. Configuring Dynamic Binding.
 - Add target interfaces to a mediation component.
 - Specify that the interfaces are dynamic.
2. Set the dynamic reference task.
 - Add the Set Dynamic reference task to your mediation path.
3. Configure dynamic references in the composite.
 - Create a component reference and specify that it is wired by implementation.
 - Wire the dynamic component reference of the mediation component to the dynamic composite reference.
4. Create and deploy composites used by dynamic binding.
 - You can create composites with service virtualization that either implement the service or pass-through to a SOAP or JMS service.

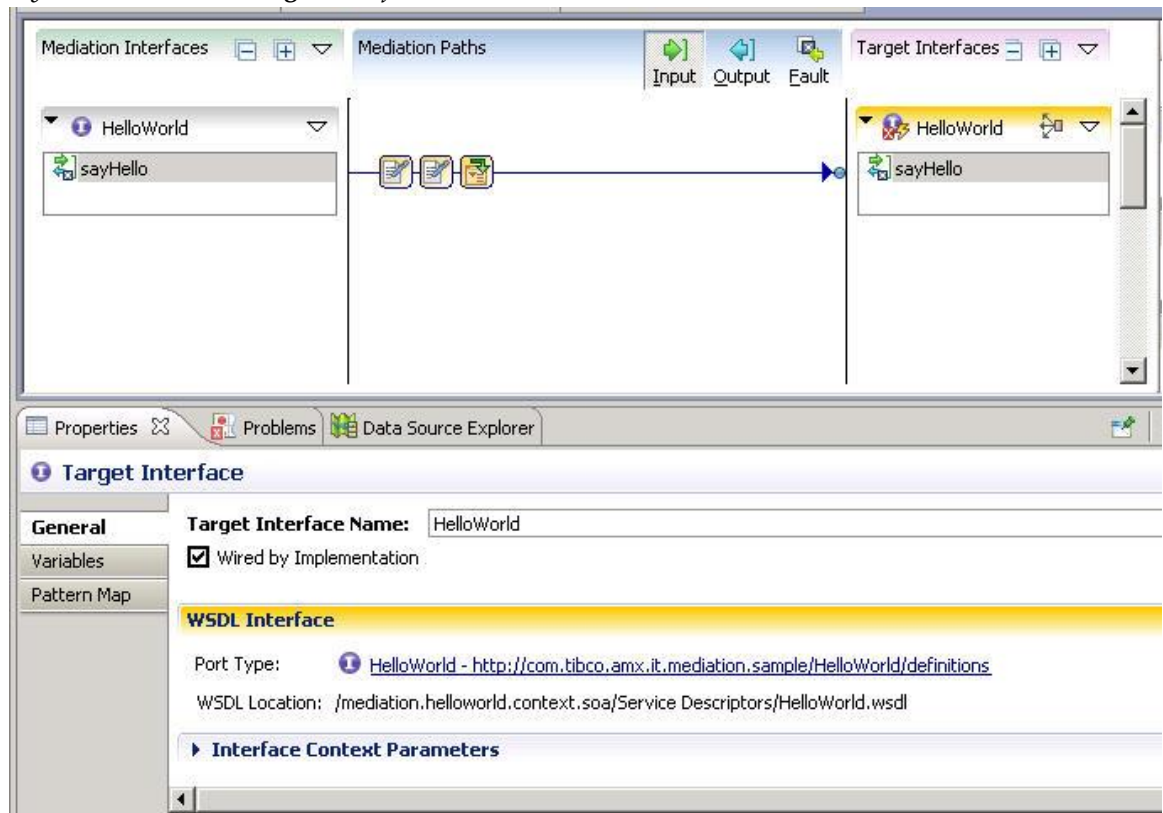
Configuring Dynamic Target Interfaces

You can specify whether a target interface is static or dynamic. By default, target interfaces are static. The target interface corresponds to a component reference that is wired to a composite reference in a composite. Dynamic target interfaces correspond to dynamic component references that are wired to dynamic composite references.

Procedure

1. Go to the General tab of the Properties view of the target interface.
2. Select the **Wired by Implementation** field.

Dynamic and static target interfaces



The title bars of the target interfaces is shaded yellow, and a lightning bolt icon is added.

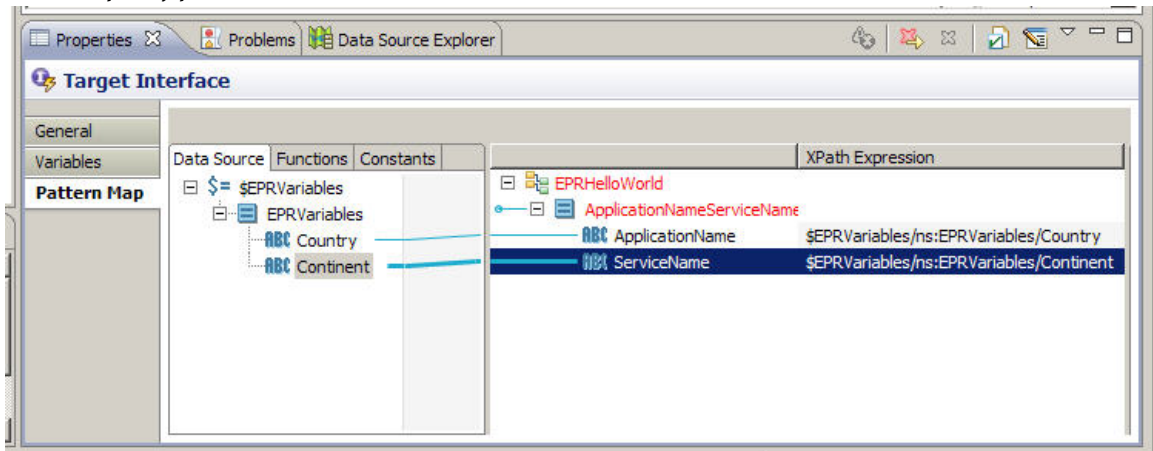
Pattern Variables Usage

Once a target interface is marked as dynamic, the application name and service name must be supplied in the mediation flow. You can optionally specify pattern variables to aid in the mapping of data in the mediation exchange to create the appropriate application name and service name.

For example, you can have six operations in your dynamic target interface. All service providers that are referenced use the same application name, and service name follows the pattern service data, where data is a variable portion of the service name. The value of data is supplied in an incoming message.

Instead of providing a mapping for each of the six operations, you can specify one pattern variable to represent the variable portion of the service name, and then create one mapping for all operations on the **Pattern Map** tab of the Properties view of the target interface. The figure below shows the **Pattern Map** tab of this example.

An example of pattern variables



To supply the value of the pattern variable, you can perform one simple mapping in a Set Dynamic Reference task on the path for each mediation operation.

See [Dynamic Reference Task Setting](#).

Dynamic Reference Task Setting

The Set Dynamic Reference task provides the values needed for resolving a service provider in a dynamic target interface.

Each Set Dynamic Reference task sets the value of the service provider for the specified dynamic target interface. The value is then used by the next item that refers to a dynamic target interface — either the end of the mediation path points to a dynamic target interface, or an Invoke task invokes an operation on a dynamic target interface.

You may need more than one Set Dynamic Reference task along a mediation path in these situations:

- The target interface is marked as dynamic and there is an Invoke task on the path configured invoke a different dynamic target interface.
- More than one Invoke tasks are on a path, and each task invokes a different dynamic target interface.
- You want to Invoke the same operation on a dynamic target interface more than once, and each time you want to set the dynamic reference to a different value.

Some use cases of the Set Dynamic Reference task are given below:

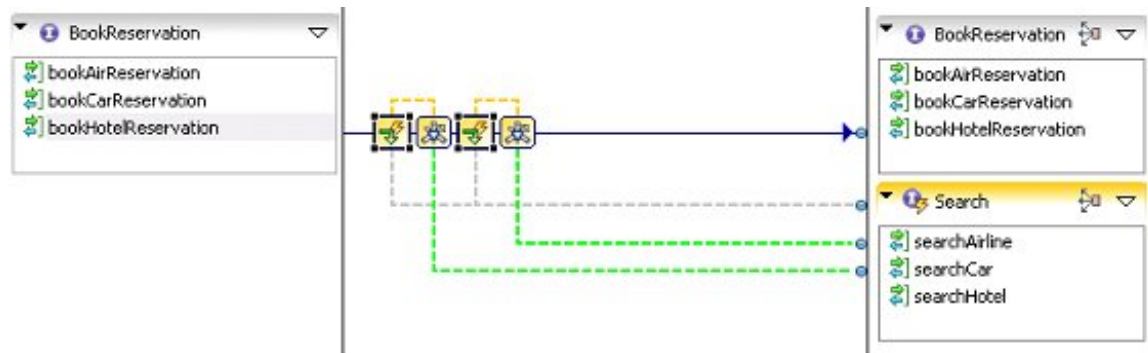
- One dynamic target interface and one Set Dynamic Reference task.



- Two dynamic target interfaces and two Set Dynamic Reference task. The first Set Dynamic Reference task sets the reference for the target operation. The second Set Dynamic Reference task sets the reference for the Invoke task.



- Two Invoke tasks, each executing different operations on the same interface. The first Set Dynamic Reference task sets the reference for the first Invoke task. The second Set Dynamic Reference task sets the reference for the second Invoke task. A different service provider can be invoked by each Invoke task.

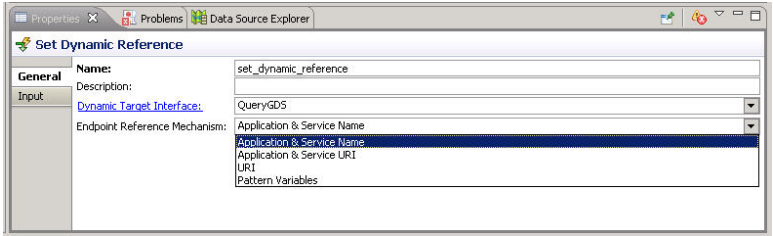


A grey hint line appears between the Set Dynamic Reference task and the corresponding dynamic target interface. A yellow hint line appears between a Set Dynamic Reference task and the corresponding Invoke task when you select a Set Dynamic Reference task in the mediation flow. The diagrams have been changed to show all hint lines, even though only the hint lines for the selected task can be viewed in the mediation editor.

The Set Dynamic Reference task is typically used on input mediation paths. It can be used on output or fault paths when an Invoke task that invokes an operation on a dynamic target interface appears on an output or fault path.

General Tab Configuration

Use the **General** tab to specify the name and description of the Set Dynamic Reference task, and to set the target interface and endpoint reference mechanism.

Name	Description
Dynamic Target Interface	Specifies the name of the dynamic target interface for which this task is supplying the service name and namespace. By default, this field is automatically set to the dynamic target interface at the other end of the path. If there is an Invoke task on the path, this field may be set by default to the first dynamic target interface in the target interface list. You might need to set this field when the default choice does not match the dynamic target interface that you want to set.
Endpoint Reference Mechanism	<p>Select the mechanism to use for setting the application and service name</p> 
Application & Service Name	Select to supply the application name and service name. This option requires two inputs for mapping on the Input tab — ApplicationName and ServiceName.
Application & Service URI	Select to supply the exact URI of the endpoint. This is useful if, for example, someone sends you the URI—you can simply copy and paste it into the ApplicationServiceURI parameter on the Input tab.
URI	Select this option to specify the URI in the Input tab.
Pattern Variables	Select to use pattern variables from the dynamic target interface. This is useful if several operations in a dynamic target interface use a similar pattern for the application name and service name. You can specify the mapping once on the dynamic target interface and use variables to supply the variable portion. The variables you create on the dynamic target interface appear in the Input tab when this option is selected.

By default, the **Endpoint Mechanism** field is set to Application & Service Name when the dynamic target interface has no pattern variables.

If the dynamic target interface has pattern variables, the **Endpoint Mechanism** field is set to Pattern Variables by default.

Input Specification

Field	Input Value
General > Endpoint Reference Mechanism > Application & Service Name	<p>The input elements for this task are <code>ApplicationName</code> and <code>ServiceName</code>. Any value you specify for these input elements override the value specified on the Pattern Map tab of the specified dynamic target interface.</p> <ul style="list-style-type: none"> • <i>ApplicationName</i> refers to the application name provided during deployment of a composite, to uniquely identify an instance of an application template. • <i>ServiceName</i> is the name of the composite service that is contained in the target composite.
General > Endpoint Reference Mechanism > Pattern Variables	<p>The pattern variables defined on the specified dynamic target interface are the input elements. This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the Pattern Map tab of the dynamic target interface.</p>
General > Endpoint Reference Mechanism > Application & Service URI	<p>The input element for the Set Dynamic Reference task is <code>ApplicationServiceURI</code>. The data type of the ApplicationServiceURI input field is a URI of the format <code>urn:amx:EnvironmentName/ApplicationName#service(ServiceName)</code></p> <p>The variables <i>EnvironmentName</i>, <i>ApplicationName</i>, and <i>ServiceName</i> refer to the environment and service that are being invoked.</p> <ul style="list-style-type: none"> • <i>EnvironmentName</i> is the name of the TIBCO ActiveMatrix environment that contains the target service.
General > Endpoint Reference Mechanism > URI	<p>The input elements for the Set Dynamic Reference task are URI.</p> <ul style="list-style-type: none"> • SOAP over HTTP • SOAP over JMS • ActiveMatrix Service Virtualization

Use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values. See [Transforming Tasks](#) for information on mapping data in the Input tag.

The content of the **Input** tab depends on which Endpoint Reference Mechanism you select on the **General** tab — Application & Service Name, Application & Service URI, URI, or Pattern Variables.

Configuring Dynamic References in Composite

Dynamic target interfaces in a mediation flow correspond to dynamic component references in mediation components that use the mediation flow as an implementation. Dynamic component references must be wired to dynamic composite references in a TIBCO ActiveMatrix composite. See *Composite Development Guide* for more information about creating and configuring composite references.

Procedure

1. Open the **General** tab of the promoted reference.
2. In the **Advanced** section, select the **Wired by Implementation** field.
The references and services must be promoted to the composite level for this setting to take effect.

Creating and Deploying Composites Used By Dynamic Binding

TIBCO ActiveMatrix resolves the application and service names provided by a component to a running application that contains the corresponding service of binding type virtualization. The composite with the corresponding service can implement a service or it can pass through to another service using the SOAP or JMS protocol.

See [Service Providers for Dynamic Composite References](#). You can create composites using the Composite Editor, or you can use the automatic mechanism in the Mediation Editor to create composite services that a dynamic target interface can use.

Procedure

1. Click the down-arrow icon in the title bar of a dynamic target interface in a mediation flow and select **Create Dynamic Provider** from the menu.
This Create Dynamic Provider dialog opens.
2. Specify the **Service Name**, the **Namespace** and the **Workspace Location** in the fields provided.
3. Click the **Browse** button next to the **Workspace location** field to locate the project and folder in your workspace where you want to place the composite.
4. Click **OK**.

Result

The provider composite created with the wizard is configured with a service with the specified name and namespace. The port type and WSDL location for the service are set to the target interface in the mediation flow.

Replying to Messages

Generate Reply and Handle Reply tasks can be used to send reply messages without invoking target operations.

In a typical mediation flow for an operation with an in-out message exchange pattern, incoming messages travel along the input path until the message is delivered to the target operation or until a fault is encountered.

In some situations, you might want to send a reply message to the consumer without invoking the target operation. For example, an operation might return the name of the target service. The mediation flow already has the target service name, so you can improve performance and return that information without additional network traffic to the target service.

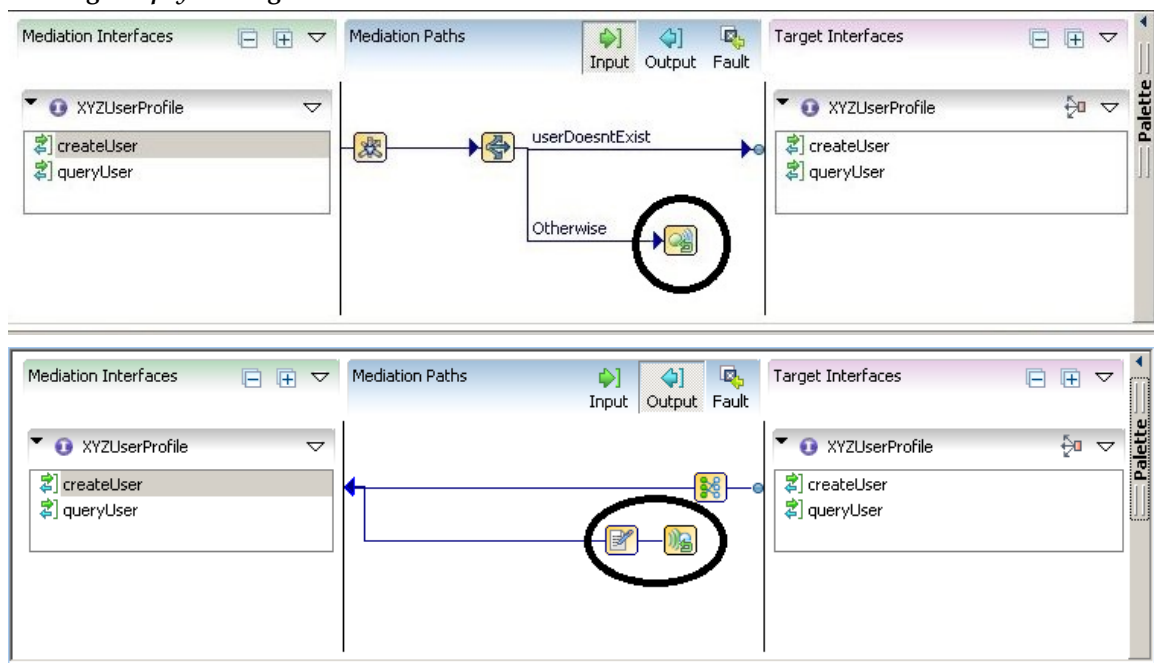
Another example is a mediation flow with a route task for processing incoming requests. Your mediation flow might return an unchangeable message for one or more routing cases. In that case, you can reply to the consumer without invoking the target service.

You can place the Generate Reply task on an input path to terminate the path and pass control to the output path of the mediation flow. You must map the output message in the **Generate Reply Input** tab, so that the output message is created in the task.

On the output path, the Handle Reply task intercepts messages from any Generate Reply tasks on the input path and starts the mediation reply path for processing the reply message before it is sent to the consumer.

The diagram shows the operation of the Generate Reply and Handle Reply tasks. In this example, a mediation flow for the `createUser` operation first invokes the `queryUser` operation to determine if the user exists. If the user does not exist, the message is delivered to the `createUser` target operation. If the user already exists, the Generate Reply task is used to return a message notifying the consumer that the user already exists.

Sending a reply message



Placing a Generate Reply task in the Input path automatically creates a mediation reply path with a Handle Reply task. The same Handle Reply task performs all Generate Reply tasks in the Input path.

The Generate Reply task terminates an input path before reaching a target operation. However, you can have more than one Generate Reply task on an input path when a route task splits the input path into

multiple sub-paths. One or more sub-paths can end in a Generate Reply task. Generate Reply tasks are executed based on how they are configured in the input flow paths. The Handle Reply task is on the output flow.

After a Generate Reply task is executed, control is passed to the Handle Reply task on the output path. One Handle Reply task accepts reply messages from any Generate Reply task on the input path. The Handle Reply task starts the mediation reply path. Optionally, you can place tasks on the mediation reply path to perform additional processing before the reply message is sent to the consumer. The Handle Reply task and the mediation reply path are automatically placed into the mediation flow when a Generate Reply task is placed on the input path.

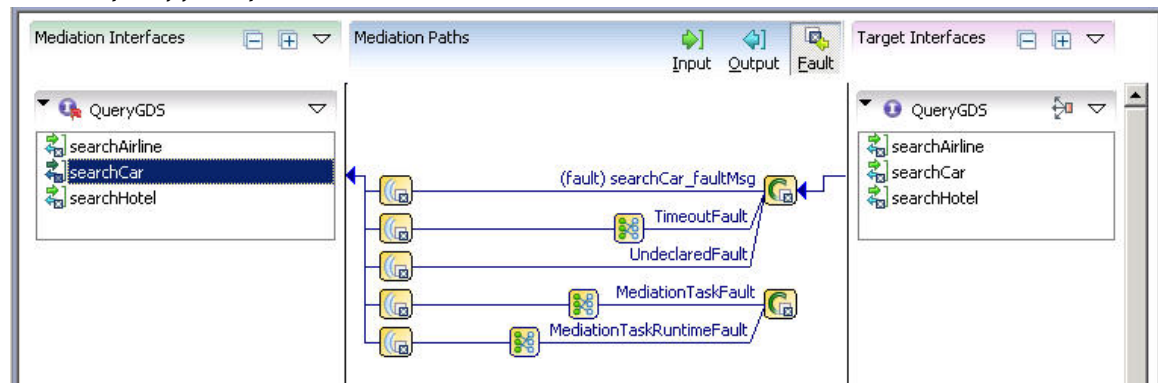
Fault Processing in a Mediation Flow

Faults are errors that can occur at any point along the mediation path. Faults are caused by the target service while processing messages. Faults can also explicitly occur during a mediation flow to specify that an unhandled case has occurred. The Mediation Flow Editor enables you to specify a fault path for processing to occur when a fault is encountered.

Invoke tasks, like any other task, can generate faults. If an operation referenced by an Invoke activity declares faults, those faults can be caught and processed in the fault path. For example, if the operation declares that it can throw *FaultA* and *FaultB*, these faults appear in the Fault Path as faults that can be caught and processed.

- Faults can occur when receiving the message and creating the mediation flow context.
- Faults can occur when executing tasks in input or output or fault paths of the mediation flow.
- Faults can occur when executing the target operation.

An example of fault path



When a fault is encountered, processing of the current path is immediately halted, and control is passed to the fault path. The fault path enables you to catch explicitly declared faults or unhandled faults. However, if a fault is encountered when receiving the message, the mediation flow has not yet started, so the fault is immediately returned to the sender and no fault processing can be done in the mediation flow.

By default, each target operation has one Catch Fault task with sub-paths.

- A sub-path for each declared fault that can be thrown by the operation.
- One sub-path to handle any undeclared faults.
- One sub-path for timeout faults.

There is also one Catch Fault task for faults that occur during processing of the mediation flow.

You can configure each Catch Fault task to have fewer sub-paths, if desired. When you remove sub-paths from a Catch Fault task, the Catch All path is automatically added to catch any faults where there is no specific sub-path for fault handling.

Each sub-path from each Catch Fault task leads to a Send Fault task. The Send Fault task sends a fault back to the original sender of the message. By default, the Send Fault task is configured to send the specific fault caught by the sub-path. You can configure the Send Fault task on either a target or mediation fault sub-path to send either a generic UndeclaredFault or one of the specific fault messages defined on the mediation operation.

When the fault sent by the Send Fault task does not match the fault caught by the sub-path, a Transform activity is required to transform the fault message into the required format. For faults on the mediation fault path, the Transform activities are added by default, but if you change the configuration of the Send Fault or Catch Fault tasks, you must provide the correct Transform task as well.

You can place mediation tasks along the sub-paths between the Catch Fault activities and Send Fault activities to perform post-fault processing before the fault is returned to the original message sender.

For more information about how to configure the [Catch Fault](#) and [Send Fault](#) tasks, see [Working with Fault Paths](#).

When faults are encountered while processing tasks in a mediation flow, the execution of the path is terminated, and the control sent to the mediation fault path. This includes faults that occur when processing takes on any of the following paths:

- Mediation Input Path
- Mediation Output Path
- Mediation Reply Path
- Mediation Target Fault Path

When a fault is encountered on the Mediation Fault Path, the path terminates and a fault is sent to the consumer.

Throwing Faults in Mediation Flows

The Throw Fault mediation task enables you to explicitly throw a fault during processing on the input path of a mediation flow.

This is useful in two situations:

- You want to deprecate a mediation operation, and therefore a fault is sent to all clients that request that operation.
- You want to specify routing cases where a fault is sent.

For example, if a loan processing application cannot process loans over \$5,000,000, then you would configure a routing case for the loan request operation to examine the loan amount and place a Throw Fault task on the sub-path for the case where the loan amount was over \$5,000,000.

The Throw Fault task enables you to browse through available service descriptors and select messages from the service to send as the fault message. You also can select which MediationTaskFault message to send. If you have more than one Throw Fault task and you want to perform specific processing for each task, configure each task to send a specific message.

Procedure

1. On the **General** tab, click **Browse** to select a service descriptor containing the fault message to send.
2. On the Select WSDL Message dialog, select the WSDL file in the Matching Resources field.
3. The Throw Fault activity is configured to throw the message, and you can navigate to the WSDL by clicking the WSDL Location field label.

Fault Paths

Fault paths enable you to specify tasks to perform when a fault is thrown.

To view the fault path for the currently selected mediation operation, use the **Show Fault Direction** button at the top of the mediation paths area of the mediation flow editor. There is one Catch Fault task for each target operation in the mediation flow, and one Catch Fault task for faults encountered while processing the mediation flow.

Each target operation in a mediation flow has a Catch Fault task that catches faults thrown by the target operation. The faults can be either explicitly defined faults in the target operation's service description, or they can be unhandled exceptions encountered during processing (for example, a `NullPointerException`).

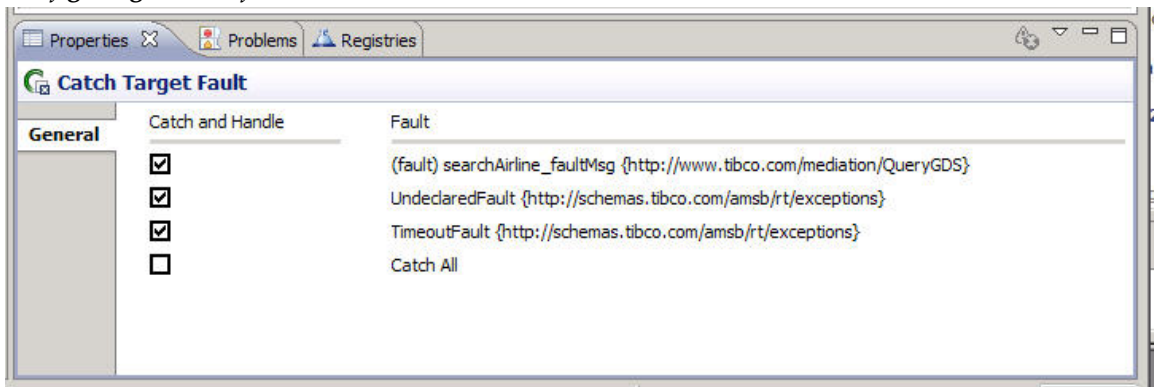
The default Catch Fault task for a target operation has sub-paths for each declared fault in the target operation and one sub-path each for a time out fault and any undeclared faults. You can place mediation tasks on each sub-path to perform any post-fault processing for each fault.

To specify the same processing for multiple faults, you can configure the Catch Fault task to have fewer sub-paths by unselecting the **Catch and Handle** field for the fault. When you eliminate one or more sub-paths, the Catch All sub-path is required, and it is automatically enabled. Any faults that do not have a defined sub-path are sent to the Catch All sub-path.

Catch Fault Configuration

The **General** tab of the Catch Fault task that allows you to configure the sub-paths for the faults to catch.

Configuring a catch fault task

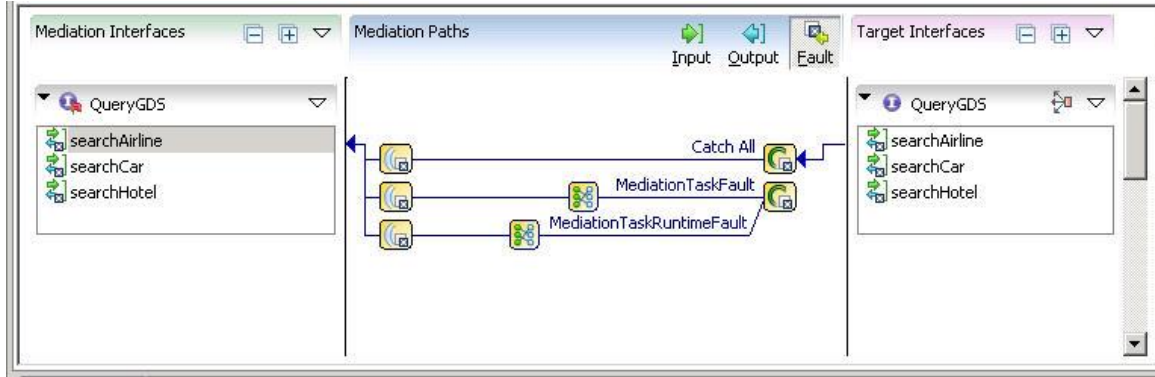


In the above example Catch fault task, one fault message is defined on the target operation named `searchAirline_faultMsg`. The Catch Fault task also has the following faults listed for all target operations:

Faults	Description
UndeclaredFault	Catches any undeclared faults encountered while executing the target operation.
Catch All	Catches all remaining faults that are not explicitly defined. By default, this option is cleared. This option becomes required and is automatically selected if you clear the Catch and Handle field for any other faults.
Timeout Fault	Catches any timeouts encountered while executing the target operation.

When you select the check box in the Catch and Handle column for a fault, the Catch All sub-path is automatically added. The following figure shows the fault path that results when the Catch and Handle check box is cleared for `searchAirline_faultMsg`, `Undeclared Fault`, and `TimeoutFault`.

Removing specific faults from the target operation fault path



Catching Faults from the Mediation Flow

One Catch Fault task catches faults encountered while processing the mediation flow.

Faults in a mediation flow can occur in the following situations:

- An explicit fault is thrown with the Throw Fault task. This task can either throw the MediationTaskFault message or it can be configured to throw a different message defined in a service descriptor in the project.
- A mediation task throws the declared MediationTaskFault fault during processing (this also applies to tasks on target fault paths).
- An undeclared exception occurs during mediation processing. In this case, the MediatinTaskRuntimeFault is thrown.

Catch fault task for the mediation flow

The screenshot shows the 'Catch Mediation Task Fault' configuration dialog. It has a 'General' tab and a 'Catch and Handle' section. The 'Catch and Handle' section has three checkboxes: 'Catch and Handle' (checked), 'MediationTaskFault' (checked), and 'MediationTaskRuntimeFault' (checked). The 'Fault' section lists the following faults: 'MediationTaskFault {http://schemas.tibco.com/amsb/rt/exceptions}', 'MediationTaskRuntimeFault {http://schemas.tibco.com/amsb/rt/exceptions}', and 'Catch All'.

The above figure shows the Catch Fault task for the mediation flow. In this example, the mediation flow has a Throw Fault task that throws the searchHotel_faultMsg fault, and the MediationTaskFault, MediationTaskRuntimeFault, and catch all options are present in all Catch Fault tasks for mediation flows.

By default, the MediationTaskFault and MediationTaskRuntimeFault sub-paths are configured with Transform tasks that transform the caught fault into an UndeclaredFault message. If you check the **Catch and Handle** field for any faults declared on the target operation, you must configure the corresponding Send Fault task and provide any required transformations by adding a Transform task to the sub-path, if necessary.

Catching All Faults

If you choose not to catch specific faults from the target operation or the mediation flow, the **Catch All** fault option remains selected. In this case, the **Fault to Send** field of the Send Fault task contains an option **Original Fault That Was Thrown**. When this option is selected, all encountered faults are passed on the caller as they occur without any transformation.

Sending Faults to the Invoker

The Send Fault task sends a fault message back to the original process that invoked the mediation operation.

You can configure the Send Fault task to specify what fault message to send:

- One of the fault messages declared on the mediation operation
- The UndeclaredFault message

Procedure

1. In the Send Fault task select **General > Fault to Send**.
2. Specify the fault message to return.
3. Once you specify the fault message to return, place a Transform task on the fault path to convert the message sent by the Catch Fault task to the format of the fault message you are returning.

Result

The message panel of the Transform task on a fault path is labeled Mediation Fault Message, and the schema of the fault message matches the schema of the message specified in the Send Fault task on the path.

Custom Mediation Tasks

Custom mediation tasks are user-defined mediation tasks written to perform specific mediation functions.

Migrating Custom Mediation Tasks

If you created and deployed custom mediation tasks in earlier versions of TIBCO ActiveMatrix, and you want to use them with this version, you must migrate them to TIBCO ActiveMatrix 3.x.

Procedure

1. Import your TIBCO ActiveMatrix 2.x model and UI plug-ins into your workspace.
2. The Model plug-in uses import packages. Replace all Required-Bundles with:


```
com.tibco.amsb.core.mediation.model.report.impl
com.tibco.amsb.core.componentType
com.tibco.amsb.core.mediationbpel.custom.task.persistence
com.tibco.amsb.core.mediationextmodel
com.tibco.amsb.core.mediationextmodel.impl
com.tibco.amsb.core.mediation.model.ext.internal
com.tibco.amsb.core.querymodel
com.tibco.amsb.core.task.model.ext
com.tibco.amsb.core.task.model.report
com.tibco.amsb.core.taskdescriptormodel
com.tibco.amsb.task.common.validation
com.tibco.amsb.core.task.validation.constraints
org.eclipse.core.runtime
org.eclipse.bpel.model
org.eclipse.emf.common
org.eclipse.emf.ecore
org.eclipse.emf.ecore.xmi
org.eclipse.emf.ecore.impl
org.eclipse.emf.validation
org.eclipse.emf.common.util
org.eclipse.emf.ecore.plugin
org.eclipse.emf.common.notify
org.eclipse.emf.common.notify.impl
org.eclipse.emf.ecore.xmi.impl
org.eclipse.wst.wsdl org.eclipse.xsd
```
3. Delete the methods `getStaticInputTypeSchemaStream()` and `getStaticOutputTypeSchemaStream()` methods from class `<TaskName>MediationTaskReport.java`.
4. Delete the class `<TaskName>MediationTaskValidator.java` after plugging out the validation code as the 3.x validation is based on EMF model constraints. The CMT developer must migrate the existing validation to EMF model constraints.
Remove its extension entry in `plugin.xml` by deleting `com.tibco.amsb.core.mediation.model.ext.MediationTaskValidationExtension` entry.

5. Update the the UI plug-ins.
 - a) Update the following dependencies from:


```
com.tibco.amsb.core.mediation.model
com.tibco.amsb.core.mediation.model.report
com.tibco.amsb.core.mediation.model.ext
```

 to


```
com.tibco.amsb.core.mediation.model.eclipse
com.tibco.amsb.core.mediation.model.report.eclipse
com.tibco.amsb.core.mediation.model.ext.eclipse
```
 - b) Add the dependency: `com.tibco.xml.mapper.schema.emfapi`.
6. Re-compile the plug-ins by doing a clean build.
7. Re-create your runtime plug-in, and port your custom code using the Mediation API.
See [Creating the Runtime Plug-in](#).
8. Re-create the DAA in the SOA project that is created. See [Deploying Custom Mediation Tasks](#).
9. Install the custom mediation task. See [Writing Custom Mediation Code](#).

Eclipse Plug-in Reference



A custom mediation task consists of three Eclipse plug-ins.

Custom mediation task plug-ins

Plug-in	Description
Model	<p>The basis of automatic code generation for the design and runtime environments. The model contains attribute-value pairs that can be used in both environments. This plug-in consists of metadata based on the Eclipse Modeling Framework (EMF).</p> <p>This plug-in is used in both the design and runtime environments.</p> <p>See Creating the Model Plug-in.</p>
UI	<p>The user interface code and icons. This plug-in has extension points for the Properties view and the Mediation Palette in the Mediation Flow Editor.</p> <p>This plug-in refers to the model and is used in the design environment.</p> <p>See Creating the UI Plug-in.</p>
Runtime	<p>The Java code that performs the mediation logic.</p> <p>This plug-in refers to the model and is used in the runtime environment.</p> <p>See Creating the Runtime Plug-in.</p>

Support Files

A custom mediation task might depend on support files such as schema files and graphic files. Schema files describe the schemas of messages, and graphic files are used as icons for the custom mediation task. The icon formats can be GIF, JPEG, or PNG formats.

Icon	Where Displayed	Recommended Dimensions
Small	Mediation Palette: When the Use Large Icons option is not selected	16 x 16 pixels
	Paths: When Small Icons is selected in the preferences	Default icon: 
large	Mediation Palette: When the Use Large Icons option is selected	32 x 32 pixels
	Paths: When Large Icons is selected in the preferences	Default icon: 



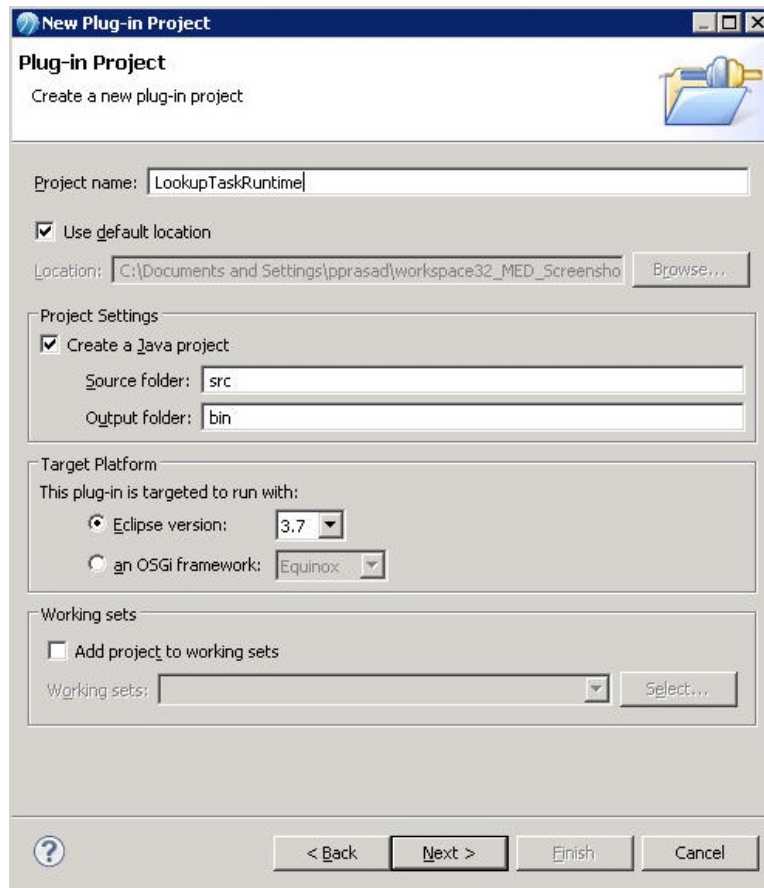
In the palette, icons appear on a light gray background. On the canvas, icons appear on a yellow gradient. For this reason, consider using a combination of hard edges (rather than anti-aliasing) and transparency when designing icons.

Creating the Model Plug-in

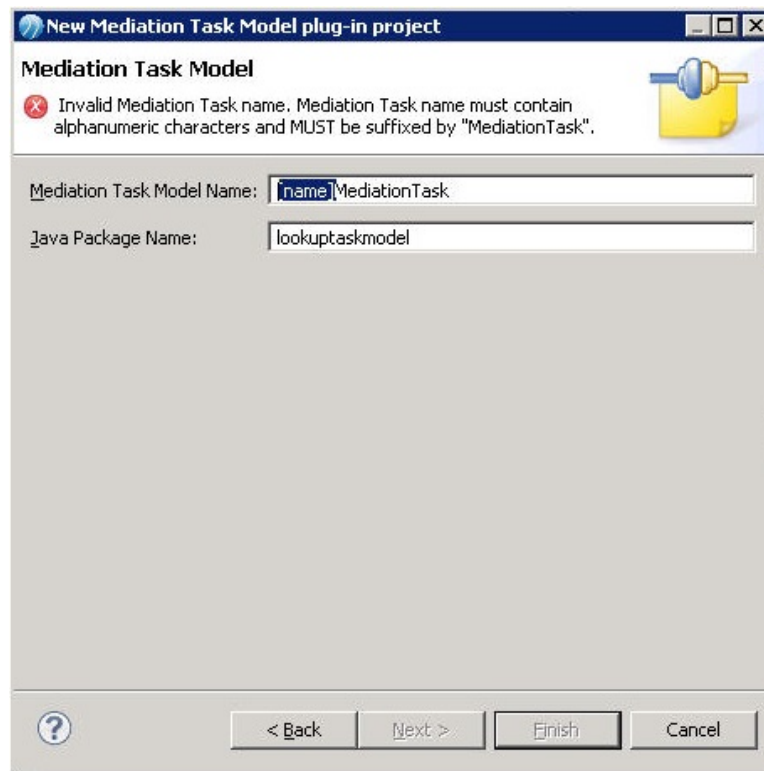
You can create the model plug-in for the custom mediation task.

Procedure

1. Run TIBCO Business Studio from the Start menu.
For example, select **Start Programs > TIBCO_Home > TIBCO Business Studio N.N > TIBCO Business Studio**.
2. Select **File > New > Project**
3. In the New Project dialog under **Plug-in Development**, select **Plug-in Project** and click **Next**.
4. Specify a name for the project that reflects the mediation task name and that identifies this as the model plug-in. For example, type `LookupTaskModel`.
5. Accept all other defaults and click **Next**.

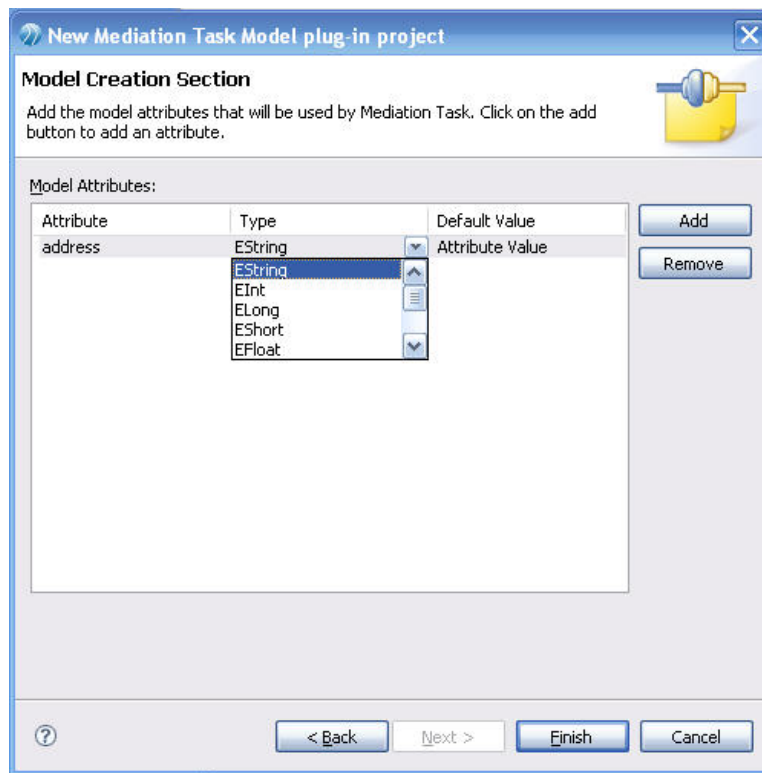


6. On the Plug-in Content page, locate the Plug-in Options group and clear these options.
 - **Generate an activator, a Java class that controls the life cycle of the plug-in.**
 - **This plug-in will make contributions to the UI.**
 7. Accept all other defaults and click **Next**.
 8. On the Templates page, select **Mediation Task Model Wizard** and click **Next**.
 9. On the Mediation Task Model page, for the **Mediation Task Model Name** replace the string within the brackets with another of your choice. This prefix will be used for the Mediation Task Names for the UI and Runtime plug-ins.
- The below figure highlights the string to replace:



10. Specify the **Java Package Name** for the model plug-in for the custom mediation task. Click **Next**.
By default the Java package name is the same as the project name.
11. (**Optional**) Select a schema element for the custom mediation task input/output in the Input/Output Schema Selection dialog.
 - a) Type the name of the schema.
 - b) Click **Browse** to see all the schemas in the workspace, or click **Create** to create a new schema.
Clicking **Create** opens the Simplified Schema Editor. See [Creating Simple Schemas](#).
12. Specify model attributes that the custom mediation task will use. Model attributes can be given values for each instance of the task, by specifying the values on the General tab in the Properties view for the task.
 - To add an attribute, click **Add**. Edit the attribute name and add default values for the attribute. Click the **Types** cell to select the attribute type.
 - To remove an attribute, highlight the row for the attribute by clicking in one of the cells on the row, and click **Remove**.

The following figure shows an example of a new model attribute:



13. Click **Finish**.
14. TIBCO Business Studio opens the Open Associated Perspective dialog, which asks if you want to open the Plug-in Development perspective.
 - Optionally, check the check box **Remember my decision**. Select **Yes**. TIBCO Business Studio opens the model plug-in and the Plug-in Development perspective.

Result

The model plug-in for the custom mediation task is created.

Creating the UI Plug-in

UI plug-in refers to the model and is used in the design environment.

Prerequisites

Before you begin, close your runtime plug-in project.

Procedure

1. Close the RT project.
2. Run TIBCO Business Studio.
3. Select **Start > All Programs > TIBCO_HOME > TIBCO Business Studio N.N > TIBCO Business Studio**.
4. Select **File > New > Project**
5. In the New Project dialog under **Plug-in Development**, select **Plug-in Project** and click **Next**.
6. Specify a name for the project that reflects the mediation task name and that identifies this as the UI plug-in — for example, *LookupTaskUI*.

7. Accept all other defaults and click **Next**.
8. On the Plug-in Content page, accept all defaults and click **Next**.
9. On the Templates page, select **Mediation Task UI Wizard** and click **Next**.
10. In the Mediation Task Model Selection Section, choose the mediation task model plug-in and click **Next**.
11. On the mediation Task UI page, the prefix that was chosen for the Mediation Task Name for the Model appears. Accept the default or specify a new one.
12. Specify the **Java Package Name** that will be used for the UI plug-in for the custom mediation task, or accept the default value. Click **Next**.
By default, the Java package name is the same as the project name.
13. Specify the location of the small icon for the custom mediation task. The location should be the complete path to the file on your local hard drive. Click **Browse** to locate and select the file.
14. Specify the location of the large icon for the custom mediation task. The location should be the complete path to the file on your local hard drive. Click **Browse** to locate and select the file.
15. Click **Finish**.

Result

The UI plug-in for the custom mediation task is created.

You can now install the custom mediation tasks. See [Installing Custom Mediation Tasks](#).



In case you see compilation errors switch the Target Platform to TIBCO ActiveMatrix SOA Studio. See *Composite Development* for information on switching the Target Platform.

Creating the Runtime Plug-in

The plug-in refers to the model and is used in the runtime environment.

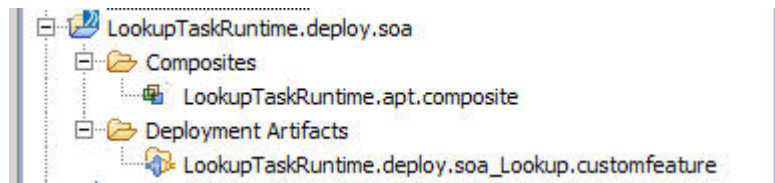
Prerequisites

Before you begin, close your UI plug-in project and your feature project.

Procedure

1. Close the feature/UI project that was created when the custom mediation tasks were installed.
2. Run TIBCO Business Studio.
3. **Start > All Programs > TIBCO_HOME > TIBCO Business Studio N.N > TIBCO Business Studio**.
4. Select **File > New > Project ...**.
5. Specify a name for the project that reflects the mediation task name and that identifies this as the runtime plug-in, for example, *LookupTaskRuntime*.
6. Accept all other defaults and click **Next**.
7. On the Plug-in Content page, locate the Plug-in Options group and deselect these options:
 - **Generate an activator, a Java class that controls the life cycle of the plug-in**
 - **This plug-in will make contributions to the UI**
8. Accept all other defaults and click **Next**.
9. On the Templates page of the wizard, select **Mediation Task Runtime Wizard** and click **Next**.
10. On the Mediation Task Model Selection Section, choose the mediation task model plug-in and click **Next**.

11. Specify the **Mediation Task Name**. This is a unique name that reflects the nature of the custom mediation task, for example, *LookupTask*.
12. Specify the **Java Package Name** for the runtime plug-in for the custom mediation task. Click **Next**.
By default, the Java package name is the same as the project name.
13. Click **Finish**.
TIBCO Business Studio opens the Open Associated Perspective dialog, which asks if you want to open the Plug-in Development perspective.
14. (Optional) Select **Remember my decision** check box and click **Yes**.
TIBCO Business Studio opens the runtime plug-in and the Plug-in Development perspective.
Along with the Runtime plug-in, a SOA Project `<runtime plug-in project name>.deploy.soa` is created as shown in the following figure:



This SOA project contains the Custom Mediation Task Extension component that refers to the runtime plug-in.

Your custom code is written in `<runtime plug-in project name>\src\<runtime plug-in project name>\<custom mediation task name>RT.java`.



Do not update or delete the generated `<runtime plug-in project name>.apl.composite`. This composite is generated for the sole purpose of packaging the custom tasks plug-ins into deployable artifacts.

If you see compilation errors, switch the Target Platform to TIBCO ActiveMatrix Runtime. See *Composite Development Guide* for information on switching the Target Platform.

Writing Custom Mediation Code

Custom mediation code performs operations on Mediation Exchange in paths, and on specific elements of the message and path contexts.

Prerequisites

Before modifying the Task EMF Model, import the required plug-ins:

Procedure

1. Make sure the target platform is set to TIBCO ActiveMatrix Runtime.
2. Select **Import > Plug-in Development > Plug-ins and Fragments**.
3. Click **Next**.
4. Make sure the check box for importing from the active target platform setting is selected.
5. Click **Next**.
6. Import the following plug-ins by selecting them in the **Plug-ins and Fragments Found** table and clicking **Add -->**:
`com.tibco.amsb.core.model`
`com.tibco.amsb.core.mediation.model.ext`

7. Click **Finish**.

Accessing Task Input/Output Schema

To access the input or output element declaration at runtime, you must initialize a mediation task report object *MediationTaskNameReport*. This object has the accessor methods to get the input or the output element declaration as *XSDElementDeclaration*.

The following code shows how to get the input or the output element schemas:

```
public void init() throws TaskLifecycleFault { }

public void destroy() throws TaskLifecycleFault { }

public N execute(final N input, final Exchange<N> exchange)
    throws TaskFault {
    <TaskName>MediationTask task = this.getContext().getTaskConfiguration();
    <TaskName>MediationTaskReport report = new <TaskName>MediationTaskReport(task);

    //Task input type as schema element declaration
    XSDElementDeclaration inputType = report.getCustomInputType();

    //Task output type as schema element declaration
    XSDElementDeclaration outputType = report.getCustomOutputType();

    return exchange.getMessageData();
}
```



The return type of the execute method in a custom mediation task's runtime class that extends *MediationTaskRT* must be an instance of the output schema defined for the task. If no output schema is defined, output defaults to message data *mediationExchange.getMessageData()*.

Modifying the Mediation Task Data

The execute method of the mediation task runtime class has *MediationExchange* and the task input as its arguments. The mediation exchange holds the mediation message and the exchange variable as a generic *Uxml* node *N*. Mediation Properties are held as strings.

The mediation message and properties constitute mediation task data.

As message data, or any data including exchange variables and contributed data, is based on generics, use XML API that is data model agnostic to process message data. For data manipulation you must use *gXML*. *TIBCO gXML* is an XML API that is based on generics and is data model agnostic.

This sample code shows processing message data:

```
public class HelloWorldRT<I, U, N extends I, A extends I, S, T, X> extends Task<I,
U, N, A, S, T, X>
{
    public void init() throws TaskLifecycleFault { }

    public void destroy() throws TaskLifecycleFault { }

    public N execute(final N input, final Exchange<N> exchange)
        throws TaskFault
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
exchange.getXMLProcessingContext();

        final GxDocumentSerializerFactory<N, S> sf = new
DocumentSerializerFactory<I, U, N, A, S, T, X>(pcx);

        // Configure for "pretty" printing.
        sf.setIndent(Boolean.TRUE);
        sf.setMethod(new QName("xml"));
        sf.setOmitXmlDeclaration(false);

        final StringWriter sw = new StringWriter();
        final GxDocumentSerializer<N> serializer = sf.newDocumentSerializer(sw);
    }
}
```

```

    if(input != null){ serializer.serialize(input); }else{ serializer.serialize
(exchange.getMessageData()); }

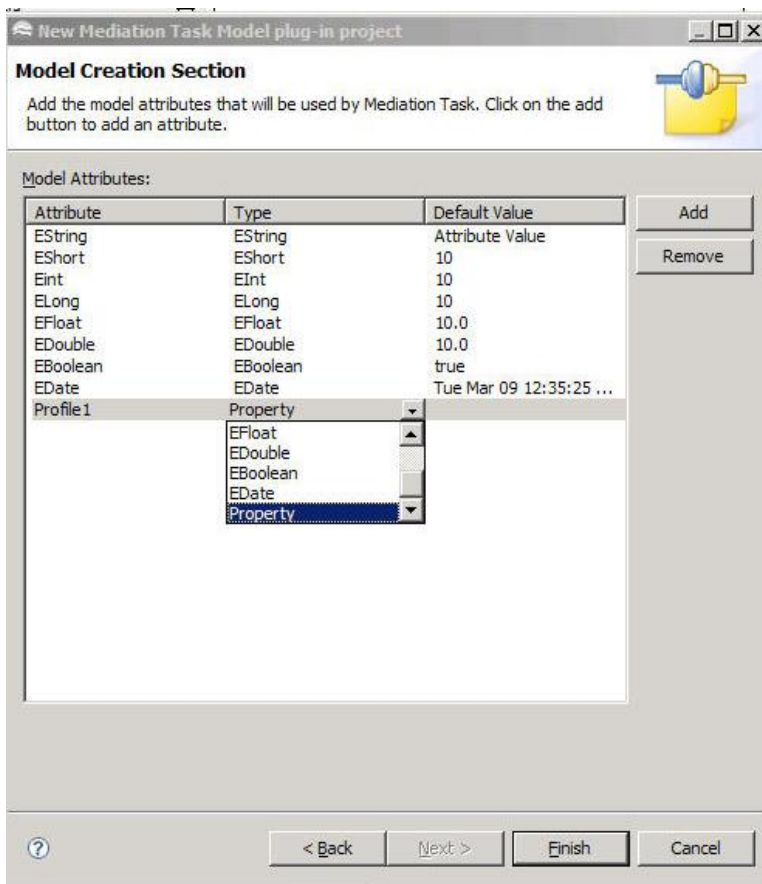
    Logger logger = LoggerFactory.getLogger>HelloWorldRT.class);
    logger.info(sw.toString());
    return exchange.getMessageData();
}
}

```

Defining Model Attributes

A user-defined mediation task can support attributes that refer to a JDBC property. Developers of the mediation task can use a property to access JDBC connections using the mediation task API provided.

To create such a task, define a Property attribute type during the model-generation phase, using the model creation page in the mediation task model wizard.



This attribute type is projected by the mediation task's user interface as a combination box that holds references to attributes defined at the mediation flow level.

Custom Mediation Task Categories

To create categories of custom mediation tasks, add the `methodgetPaletteGroup` method to the MediationTask UI factory class as shown in this example:

```

public String getPaletteGroup(){
    return "Samples";
}

```

This example creates the category Samples.

Thrown Faults

The runtime class for a custom mediation task has an execute method that throws the fault TaskFault. A developer of a custom mediation task can throw this fault explicitly.

Runtime Exceptions

Path	Exception Handling
Input	Path control is transferred to the Catch Mediation Fault task in the fault path. A Send Fault task in that path sends a message to the service consumer.
Output	Path control is transferred to the Catch Mediation Fault task in the fault path. A Send Fault task in that path sends a message to the service consumer.
Fault	A fault message is sent to the service consumer.

Installing Custom Mediation Tasks

To make your custom mediation tasks available in the Mediation Flow Editor, you must first install and deploy the plug-ins.

Procedure

1. Create a feature project.
Specify the plug-in to package into the new feature.
See **Supplemental Eclipse Help > Plug-in Development Environment Guide > Reference > Wizards and Dialogs > New Project Creation Wizards** for more information.
2. Export the feature project.
Make sure you check the check box for the Generate metadata repository option.
See **Supplemental Eclipse Help > Plug-in Development Environment Guide > Wizards and Dialogs > Export Wizards Feature Export** for more information.
3. Install the feature using **Help > Install New Software...**
Specify the location where you exported the feature project. Clear the check box for the **Group items by category** option which lists the feature project.
The custom task is ready for use and can be accessed from the palette.

Deploying Custom Mediation Tasks

After the file `<runtime plug-in project name>\src\<runtime plug-in project name>\<MediationTaskName>rt.java` is updated with the custom code, the deployable artifacts can be generated.

Procedure

1. Make sure the Target Platform points to ActiveMatrix Runtime.
See *Composite Development Guide* for information on switching the Target Platform.
2. Verify that the Model and Runtime plug-ins have no compilation errors.
3. In the Project Explorer pane, expand the `<runtime plug-in project name>.deploy.soa` project.

4. Expand the Composites folder.
5. Right-click `<runtime plug-in project name>.apt.composite`, and click **Create DAA**.

Result

The Create Deployment Archive wizard is invoked.

Refer to *Composite Development Guide* for more information on using this wizard. Deploy the DAA that packages the custom mediation task Runtime plug-ins before deploying the mediation application that uses the custom task.

Refer to *Administration Guide* for information on uploading and deploying the deployment application archive (.daa).

Testing Custom Mediation Tasks

You can test the custom mediation task in RAD by creating a Run As/Debug As configuration

Procedure

1. Add one of the following to the Functions list along with the main composite:

- A composite generated by the Custom Mediation Task wizards to the list.
- A DAA created from the composite.

Make sure that the composite or DAA that holds Custom Mediation Task is at the top of the list of Composite/DAA(s), before the SOA DAA/Composite.

2. Select **Apply** and **Run/Debug**.

Reference

Reference describes the configuration tabs for tasks and resources used in mediation flows. They are organized topically.

Catch Fault



The Catch Fault task specifies the faults to catch from a target operation or a mediation flow. Catch Fault tasks appear automatically in Fault paths. Catch Fault tasks do not appear in the palette, and cannot be added manually.

See [Fault Processing in a Mediation Flow](#).

Use the **General** tab to select or clear specific faults to catch and handle. Selecting specific faults to catch creates a sub-path for each selected fault so that you can specify processing to perform for that fault before the fault is returned to the original environment.

Select the box in the **Catch and Handle** column for the fault you want to catch. The **Fault** column provides a number of fault types

Item in Fault Column	Description
Declared Fault Message	The target operation, the Throw Fault task, and the Invoke Operation task can throw a declared fault message. The content and structure of the message varies, depending upon its declaration in the WSDL file.
UndeclaredFault	An undeclared fault that occurs while invoking the target operation returns this fault message.
UndeclaredFault	A declared fault that is thrown by one of the tasks in the mediation operation.
MediationTaskRuntimeFault	An undeclared fault that is thrown by one of the tasks in the mediation operation.
TimeoutFault	The TimeoutFault is returned when the invoked operation does not return in a specified time. The timeout value is configurable in the composite application.
Catch All	This item is always present and is selected when one or more other faults in the list are cleared. This item corresponds to the path for any faults that are not explicitly handled by other fault paths.

End Mediation



The End Mediation task ends a one-way (in-only) or a Request-Response (in-out) message exchange pattern operation.

One-way operations provide a way for service consumers to initiate operations for which they won't receive a response—the End Mediation task is an orderly way to end the mediation execution. For

example, you can log the operation's input data using the Log task and then terminate the input path of the mediation operation with an End task.

The End Mediation task can be also configured for both in-only operation and in-out operation to signal the framework to redeliver the request message or stop re-delivery of the request message.

The mediation input path of a one-way message exchange pattern operation can contain other mediation tasks before terminating with the End Mediation task. However, if any of the other tasks in the mediation input path produces a fault at run-time, this will terminate the execution of the mediation input path and transfer control to the mediation fault path. No reply is sent to the consumer, because the fault path also terminates with an End Mediation task.

You can mediate a one-way (in-only) message exchange pattern operation to a request-response (in-out) target operation. Although the mediation input path operation in this case is similar to that of a mediation flow containing a one-way operation to a request-response target operation, the behavior in the output and fault paths are different.

When mediating a one-way operation to a request-response operation, the target operation can either return a reply or throw a fault; Mediation Flow automatically terminates both with an End Mediation task:

- If the target operation returns a reply, the output path is executed and the path is terminated by the End Mediation task without sending a response to the requestor.
- If the target operation returns a fault, the target fault path is executed and the path is terminated by the End Mediation task without sending a fault to the requestor.




For a in-only message exchange pattern operation, if the end task is configured with either a Redeliver Message or Stop redeliver message option, an intent type of either At Least Once or One Way Transaction has to be defined for the mediation interface.

For a in-out message exchange pattern operation, if the end task is configured with either a Redeliver Message or Stop Redeliver Message option, an intent type of At Least Once has to be defined for the mediation interface.



When mediating one-way operations to request-response target operations, it is good practice to set a Log task to capture the response message on the output and fault paths, before the path execution stops at the End Mediation task.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Field	Description
End Type	<p>Specifies how the End Mediation task exits.</p> <ul style="list-style-type: none"> • Normal stops the mediation flow immediately. • Redeliver Message  redelivers the message that initiated the mediation flow, re-executing the entire mediation flow. • Stop-Redeliver Message  stops the redelivery of messages. • Signal an Exception  generates an exception without enforcing an intent type of either At Least Once or One Way Transaction on the mediation flow and component.

Generate Reply



The Generate Reply task is used to create a reply to a mediation operation without passing the flow of control on to a target operation.

The Generate Reply task terminates an input path and passes control to the [Handle Reply](#) task on the output path.

See [Replying to Messages](#) for more information on the Generate Reply task.

General Tab

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Input Tab

The **Input** tab is a mapping panel that you use to map the mediation exchange to the output message of the operation. See [Transforming Tasks](#) for more information about using a mapping panel.

Handle Reply



The Handle Reply is the start of the mediation reply path for handling reply messages created by any Generate Reply task on the input path.

A Handle Reply task appears automatically in the output path when a Generate Reply task is placed on the input path. Handle Reply tasks do not appear in the palette, and you cannot add these tasks manually.

See [Replying to Messages](#) for more information about the Handle Reply task.

General Tab

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hovers the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Invoke Operation



The Invoke Operation task enables you to invoke an operation of an interface in the target interface list during processing of an input, output, or fault path. The operation can be one-way or request-reply. If the operation is request-reply, the reply message is stored in the mediation exchange for use by subsequent tasks in the mediation path.

General Tab

See [Invoking an Operation](#) for more information about the Invoke Operation task.

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Field	Description
Target Operation	The operation to invoke. The drop-down list is populated with all operations from the interfaces in the target interface list of the mediation flow.

Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transforming Tasks](#) for more information about using a mapping panel.

Field	Description
Task Input	<p>A complex element containing the input message for the invoked operation. The structure of the sub-elements depends on the structure of the input message for the operation.</p> <p>You can also can input the required value for fields directly into the input schema.</p>

Output Tab

The **Output** tab contains a static tree representation of the reply message schema for the invoked operation. Subsequent tasks in the mediation flow will have access to the reply message. The reply message is stored in an element within the mediation exchange whose root is named the same as the Invoke Operation task name specified on the **General** tab.

Subsequent tasks also have access to the message context properties in the reply message. See [Working with Message Context Properties](#).

If the message exchange pattern for the operation is one-way, the output is null.

Log



The Log task sends information from the mediation flow context to the log. The Log task can be placed on an input, output, or fault path.

General Tab

For more information about the Log task, see [Logging Mediation Exchange Information](#). For information about configuring the log, see [Configuring a Log Task](#).

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Field	Description
Log Role	Select the role for log messages—INFO, WARN, DEBUG, or ERROR. Each is handled separately, and each has its own log.
Use Transform Data	<p>Select this option to display the Log tab as a mapping panel that shows the schema with the elements message, messageID, and role, so that you can build custom log messages. See Information for Custom Log Messages.</p> <p>If this option is not set, the Log tab displays information from the mediation exchange for you to specify which information to send to the log file. See Information for Standard Log Messages.</p> <p>The Use Transform Data option is cleared by default.</p>

Log Tab

The appearance of the Log tab depends on whether the option Use Transform Data is selected on the General tab.

- If Use Transform Data is clear (the default), the Log tab shows top-level message information, from which you choose the information to log. See [Information for Standard Log Messages](#).
- If Use Transform Data is selected, the Log tab appears as a mapping panel so that you can build custom log messages. See [Information for Custom Log Messages](#).

Information for Standard Log Messages

If the **Use Transform Data** option is not selected in the **General** tab, you use the Log tab to specify what top-level information from the mediation exchange to send to the log file.

Item	Description
Log All Items	Selects all sub-items on this tab and sends all information in the mediation flow context to the log.
Mediation Flow Properties	<p>The properties defined for the mediation flow. These properties can be defined on the Properties tab of the mediation flow.</p> <p>You can select the parent item to send all mediation flow properties to the log, or you can select individual properties to send the properties to the log.</p>
Mediation Flow Context	<p>Logs message context such as component and mediation flow name information, if the Mediation Flow Context option is set on the Advanced tab of the mediation operation Properties view.</p> <p>See Working with Message Context Properties for information about the Mediation Flow Context option.</p>
Message Context	<p>Logs all message context information. The message context includes information about the transport used for the message and the security context for the message.</p> <p>You can optionally select either the transport or security information if you do not want the entire message context sent to the log.</p>

Item	Description
Message Data	The content of the message.
Contributed Data	Some mediation tasks, such as Transform or custom tasks, can contribute additional data items to the mediation exchange. Each contributed data item is named for the task that contributes the data. You can send any contributed data item to the log.
Exchange Variable	You can send exchange variable information to the log if you have specified an exchange variable on the mediation operation, and have set it using the Set Exchange Variable task.

Information for Custom Log Messages

If the **Use Transform Data** option is selected on the **General** tab, the Log tab is a mapping panel, where you can map mediation information to build custom log tasks.

Field	Description
message	<p>Specify the data from the mediation exchange to log.</p> <p>You can log any data available in the mapper—the message element allows logging of a simple message, and also allows mapping XML documents in a serialized text form.</p> <p>If the Mediation Flow Context option is set on the Advanced tab of the mediation operation's Properties view, you can map message context information to the message element. See Working with Message Context Properties for information about the Mediation Flow Context option.</p>
messageID	<p>Optionally specify a message ID value to be included as part of the message that is being logged. The message ID consists of two elements, name and code. The name element is a string type and the code element is integer type.</p> <p>At run-time, the value in the name element and the value in the code element are combined to form a message ID that has the syntax name-code. For example, if the name element contains the value Mail and the code element contains the value 1000 then the message ID will be Mail-1000. However, if you only provide the value for the name element, a default value of 0 will be used for the code element. Similarly, if you only provide the value for code element, the default value for the name element will be AMSB.LogTask.</p>
role	<p>Optionally specify a logging-level role for run-time.</p> <p>Values can be <i>info</i>, <i>warn</i>, <i>debug</i>, or <i>error</i>. Values are not case-sensitive.</p> <p>If you map to this role, the value you give its property overrides the Log Role setting in the General tab.</p>

Parse XML

The Parse XML task is used when you have an XML document stored in a string or binary field.

This task produces a tree representation of the XML that can be used by subsequent tasks in the mediation flow. This task can be paired with the Render XML task to convert the parsed XML back into a string or binary field for transmission within a message.

XML documents are sometimes stored in string or binary fields to improve the performance of message transmission or for other reasons. You may want to view or manipulate the data within the document then replace the document in the message before transmission to a target operation or mediation operation. Also, the target or mediation operation in your mediation flow may expect to receive all or a subset of the fields within the document.

To parse an XML document, you must provide the schema definition for the data. The schema definition must be stored in an XSD within your project. You can use an existing XSD, create an XSD with the XSD editor within TIBCO Business Studio, or you can use your own XSD editor plug-in. See the *Eclipse XSD Developer Guide* for more information about the XSD editor within TIBCO Business Studio.

The output of the Parse XML task is placed into the contributed data portion of the mediation exchange. An element with the same name as the Parse XML task is placed into the mediation exchange. The XSD specified in the **Output Schema** field determines the structure of the element.

The Parse XML task can be placed on an input, output, or fault path.

General Tab

Field	Description
Name	<p>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</p> <p>This is also the name of the element in the mediation exchange that stores the output of this task.</p>
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Output Schema	An XSD stored in the workspace that describes the structure of the XML document you want to parse. The output of this task is a parsed XML tree containing the data in the XML document supplied in the Input tab. This output schema is the structure of the element added to the mediation exchange containing the output of this task.

Field	Description
Binary Input	<p>Select this box when the XML document is stored in binary format instead of text format. When this box is not selected, the XML document is expected to be text.</p> <p>This field controls the input element on the Input tab of this task. When you do not select this field, the input element is a string named <i>xmlString</i>. When you select this field, the input element is a binary element named <i>xmlBinary</i>.</p>
Validate Input	<p>Select this box to enable schema validation of the task input.</p> <p>If you select this box and the schema validation fails, the error results in a mediation task fault.</p> <p>If you do not select this box, validation is not performed. A fault is thrown only if a parse error occurs</p>

Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transforming Tasks](#) for more information about using the mapping panel.

Field	Description
TaskInput	A complex element to hold the input for this task. The sub-element of this element is the XML document that you want to parse. The Binary Input field on the General tab controls which of the following elements appear.
ParseXmlStringInput	Appears when the Binary Input field on the General tab is not selected. Map this element to a string element in the Mediation Exchange that holds the XML document you want to parse.
ParseXmlBinaryInput	Appears when the Binary Input field on the General tab is selected. Map this element to a binary element in the Mediation Exchange that holds the XML document you want to parse.

Output Tab

The **Output** tab is a read-only display of the output schema for this task.

Query Database



The Query Database task is used to construct a SQL SELECT statement query to a database. This task is useful for performing basic queries for looking up information stored in a database table that will be used in the mediation flow.


General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow. This is also the name of the element in the mediation exchange that stores the output of this task.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Shared Resource Property	Use the drop-down list on the field to select the available shared resource property. Shared resource properties define database connections that are used to perform the query. Shared resource properties are defined on the Shared Resource Properties tab of the mediation flow.
Max Row Count	The maximum number of rows to retrieve. The default value is 1. Specify a positive integer, or use the drop-down menu on the field to select Unlimited to return an unlimited number of rows.
Query Timeout (sec)	The timeout for the query.

Query Tab

You use the **Query** tab to define the SELECT statement for the query.

Click the connection button  to test the connection and to verify the table and column data. Clicking this button opens a connection, if you have specified a JDBC property in the mediation flow **Properties** tab, and compares the table and column data with the metadata from the database. If the connection is not successful, an error notifies you of the reason.

Three lists enable you to select tables, input data, and output columns for use in the WHERE clause of your SELECT statement

Input data is used in the WHERE clause of your SELECT statement. Use the add (+) and delete (x) buttons to the right of each list to add and delete items from each list.

- When a database connection is present and valid, the + buttons display information from the database for selecting tables and output columns.
- When no database connection is present, the + buttons allow you to add items to each list, but you must name each item and specify a type if necessary.

Clicking the + and x buttons on the Input table attempt an automatic update of the WHERE condition. If you have modified the WHERE condition, the delete might not update it and you must fix it manually.

Use the **Where Condition** field on the **Query** tab to edit the WHERE clause of the query. You can add an input variable to a condition by typing a question mark (?) in the condition. Each input variable appears in the mapper panel on the **Input** tab, and you can supply data from the mediation exchange for the input variable. For example, if you want to create a condition to look up a zip code supplied in the input message, you can add the condition `table.ZIP = ?`. When you add a question mark into the WHERE clause, an input variable appears in the Input Data list. Supply a name for the input variable, then data from the mediation exchange can be mapped to the input variable.

Table join conditions are never automatically added to the WHERE clause, so you must manually edit the WHERE clause to specify any join conditions for your query.

The **SQL Statement** field displays a read-only version of the query you have specified. The following table lists the supported SQL types and how they map to XML. Note that length parameters are stripped from the SQP Type, and only the base type is used in the mapping — for example, `char(12)` becomes `char`.

Supported SQL types and their mapping to XML

SQL/92 Data Types	XML Type Equivalent
TINYINT	short
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	float
DOUBLE	double
CHAR	string
VARCHAR	string
NCHAR	string (multi-byte)
NVARCHAR2	string (multi-byte)
DATE	date

SQL/92 Data Types	XML Type Equivalent
TIME	time
TIMESTAMP	dateTime

Vendor-specific types are cast to string. You can enable the mapper to automatically recognize these types in one of these ways:

- Force vendor-specific types to a compatible XML type using the mapper cast.
- Override the type that is retrieved from the database for the column to a similar SQL/92 type.



Binary or other complex data types such as JAVA_OBJECT are not supported.

Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transform Tasks](#) for more information about using a mapping panel.

Field	Description
InputValues	<p>A complex element to hold the input for this task. The sub-elements of this element are the input variables defined on the Query tab. Each input variable corresponds to a question mark (?) that appears in the WHERE clause of the query.</p> <p>Map values from the mediation exchange to fields in the input schema to supply values for the input variables of the query.</p>

Output Tab

The **Output** tab is a read-only display of the output schema for this task. The output schema is determined by the output columns selected on the **Query** tab.

Test Tab

The **Test** tab is used to test the database query. You must have a valid JDBC template associated with the JDBC property used by this task. The JDBC resource template is used only in the design environment.

You can use a custom JDBC driver to test the database query. For information about configuring a custom JDBC driver, see *Composite Development Guide*.



It is important for you to ensure that the JDBC resource template you use for testing in the design environment connects to a database that is similar to the database used when the project is put into production.

Render XML



The Render XML task takes an XML tree for a specified schema and converts it to a string or binary element that contains the XML document. This task can be paired with the Parse XML task to convert the parsed XML back into a string or binary field for transmission within a message.

XML documents are sometimes stored in string or binary fields to improve the performance of message transmission or for other reasons. You may want to view or manipulate the data within the document then replace the document in the message before transmission to a target operation or mediation operation. Also, the target or mediation operation in your mediation flow may expect to receive all or a subset of the fields within the document.

To render an XML document, you must provide the schema definition for the data. The schema definition must be stored in an XSD within your project. You can use an existing XSD, create an XSD with the XSD editor within TIBCO Business Studio, or you can use your own XSD editor plug-in. See the *Eclipse XSD Developer Guide* for more information about the XSD editor within TIBCO Business Studio.

The output of the Render XML task is placed into the contributed data portion of the mediation exchange. An element with the same name as the Render XML task is placed into the mediation exchange. The contents of the element is either a string or binary element containing the XML document.

The Render XML task can be placed on an input, output, or fault path.

General Tab

Use the General tab to specify a name, description, and input schema for the task. If the XML document will be stored in binary format instead of text format, you can specify that on the General Tab.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow. This is also the name of the element in the mediation exchange that stores the output of this task.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Input Schema	An XSD stored in the project that describes the structure of the XML document you want to render. The specified schema is used to determine the input schema for this task.
Binary Output	Select this box when the XML document will be stored in binary format instead of text format. When this box is cleared, the XML document is stored in a text field.
Encoding	Use this field to specify the character encoding used to render the XML string. A list of encodings is provided. If the field is empty the default system encoding is used.

Field	Description
Validate Output	<p>Select this box to enable schema validation of the task output.</p> <p>If this box is selected and the schema validation fails, the error results in a mediation task fault.</p> <p>If this box is not selected, validation is not performed; a fault is thrown only if a parse error occurs.</p>

Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transform Tasks](#) for more information about using a mapping panel.

Field	Description
TaskInput	<p>A complex element to hold the input for this task. The sub-element of this element is the schema specified in the Input Schema field on the General tab.</p> <p>Map values from the mediation exchange to fields in the input schema to create the XML document.</p>

Output Tab

The **Output** tab is a read-only display of the output schema for this task.

Route Task



The Route task sends messages to a specific destination based on specified conditions. Data from the mediation flow context, such as the security information or message body, can be used to specify the conditions of the route.

You can only introduce the route in the input path. The response (output or fault) always returns to the original requester—that is, to the same mediation operation.

Route tasks send each incoming message to a single destination based on which route case evaluates to true, or to a single destination designated as otherwise if none of the cases evaluate to true.

You can use multiple, nested Route tasks to send a single message to a target in several different ways, based on the routing cases, conditions, and variables you set for each task in the **Decision** tab.

You can configure multiple routes in an input flow, nesting them to any depth, and you can place mediation tasks on flow paths before or after any route task. This enables users to decide which tasks are executed in common and which are executed only for specific route cases.

See [Routing Messages in a Mediation Flow](#) for more information about the Route task.





General Tab

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Decision Tab

Use the Decision tab to create routing cases, routing conditions, and variables to hold data that will be evaluated in the routing conditions. The Decision tab has a toolbar for adding and deleting cases and conditions.

Toolbar Icon	Description
 Add Case	<p>Adds a routing case to the table on this tab. A new case appears in the table on this tab with a default name, and the case is drawn in the mediation flow diagram.</p> <p>By default, new cases created with this icon point to an error icon until a Target Service/Operation is specified for the case.</p>
 Delete Case	<p>Deletes the selected routing case.</p>
 Add Variable	<p>Adds a variable to use in routing conditions. Clicking this icon opens a dialog that enables you to specify the name and data type of the variable. These datatypes are available:</p> <ul style="list-style-type: none"> • string • integer • boolean • date • time • float <p>Each variable appears as a column between the Case column and the Target Service/Operation column.</p>
 Delete Variable	<p>Opens a dialog for you to select the variable to delete.</p>

The Decision tab includes a table containing all of the routing cases.

Column	Description
Case	Name of the routing case. Click in the cell to edit the name.
Variable List	<p>Variables created with the Add Variable icon appear as columns in this table. You must specify a comparison operator and a constant to compare for each variable. Click the cell to select the comparison operator from a drop-down list and edit the value of the constant in the text field.</p> <p>You can specify comparison operators:</p> <ul style="list-style-type: none"> • = (equal) • != (not equal) • < (less than) • <= (less than or equal) • > (greater than) • >= (greater than or equal) <p>All variable conditions that you specify for each case must evaluate to true for the case to evaluate to true.</p>
Target Service/Operation	The name of the Target Service and target operation that is the destination for this case. If you drag the path for the case to a target operation in the mediation flow, this field is automatically set to the correct value. You can also click this field to either type or select the target operation.

At the bottom of the Decision tab is the configuration for the Otherwise case for the route. The Otherwise case is taken when all other cases evaluate to false. You can specify the target operation for this case in the Target Service/Operation field.

Also at the bottom of the Decision tab is a drop-down list of choices for setting the case target to a specific type of mediation task. For example, selecting Throw Fault sets the target to a new Throw Fault task.

Targets you can specify are:

- Targeted operations that are not already targeted
- Generate Reply, Throw Fault for mediation tasks
- End Mediation for one-way (in-only) operations.
- Route tasks and XPath Route tasks, which enables you to build nested routing structures.



Any change you make to a nested routing structure replaces the entire nested structure.

Input Tab

Use the **Input** tab to map data from the mediation exchange into the list of variables that you have created for the Route task. See [Transform Tasks](#) for a complete description of how to perform mapping.

Send Fault



The Send Fault task returns a fault message to the original process that invoked the mediation task. Send Fault tasks appear automatically in Fault paths. Send Fault tasks do not appear in the palette, and you cannot add these tasks manually.

General Tab

See [Fault Processing in a Mediation Flow](#) for more information about fault processing.

Use the Fault tab to specify the fault to send to the original environment.

Field	Description
Fault to Send	A drop-down list of declared fault messages on the mediation operation. You can also choose to send the <code>UndeclaredFault</code> message.

Set Context



The Set Context task provides a way to set the values for the message context properties of the target operation's input message and the message context properties of the mediation operation's output message.

This allows the mediation path to set the message context data (such as HTTP header or JMS user properties) for the output message of the mediation operation and the input message of the target operation.

The schema that appears on the Set Context task is configured in the Properties view of the mediation operation or target operation. On the Advanced tab, you can set the field Message Context Properties (outbound) of the mediation operation, or the Message Context Properties (inbound) field of the target operation.

See [Working with Message Context Properties](#).

General Tab

Use the General tab to specify a name and description for the task, and to identify the operation for which to set the context.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Field	Description
Operation	<p>The operation for the Set Context task. The drop-down list is populated with all operations from the interfaces in the target interface list of the mediation flow.</p> <p>If the path is leading to the mediation Operation (for example, the Output path), you can select the mediation operation and set the context properties of the mediation output message. The Operation field identifies this case by identifying the interface. For example, <code>[Mediation Interface]:HelloWorld/sayHello</code>.</p>

Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See [Transform Tasks](#) for more information about using a mapping panel.

The input context for the target operation appears in the right side of the mapper. The task input structure provides context properties.

- Of the operation's outbound message, if a mediation operation is selected in the **General** tab.
- Of the operation's inbound context message, if a target operation is selected in the **General** tab.

Set Dynamic Reference



The Set Dynamic Reference task provides the values needed for resolving a service provider in a dynamic target interface.

Each Set Dynamic Reference task sets the value of the service provider for the specified dynamic target interface—either the end of the mediation path points to a dynamic target interface, or an Invoke task invokes an operation on a dynamic target interface.

General Tab

Use the General tab to specify a name and description of the Set Dynamic Reference task, and to set the target interface and endpoint reference mechanism.

Field	Description
Name	<p>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</p>
Description	<p>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</p>

Field	Description
Dynamic Target Interface	<p>The name of the dynamic target interface for which this task is supplying the service name and namespace.</p> <p>By default, this field is automatically set to the dynamic target interface at the other end of the path. If an Invoke task is on the path, this field may be set by default to the first dynamic target interface in the target interface list. You might need to set this field when the default choice does not match the dynamic target interface that you want to set.</p> <p>The name of the dynamic target interface for which this task is supplying the application and service name. By default, this field is set to the dynamic target interface at the other end of the path. If there is an Invoke task on the path, this field can be set by default to the first dynamic target interface in the target interface list. You might need to set this field when the default choice does not match the dynamic target interface that you want to set.</p>
Endpoint Reference Mechanism	<p>Select an option for the mechanism to use for setting the application and service name:</p> <ul style="list-style-type: none"> • Application & Service Name: Select to supply the application name and service name. This option requires two inputs for mapping on the Input tab—ApplicationName and ServiceName. • Application & Service URI: Select to supply the exact URI of the endpoint. This is useful if, for example, someone sends you the URI—you simply copy and paste it into the ApplicationServiceURI parameter on the Input tab. • URI: Select this option to specify a URI. • Pattern Variables: Select to use pattern variables from the dynamic target interface. This is useful if several operations in a dynamic target interface use a similar pattern for the application name and service name. You can specify the mapping once on the dynamic target interface and use variables to supply the variable portion. The variables you create on the dynamic target interface appear in the Input tab when this option is selected. <p>By default, the Endpoint Mechanism field is set to Application & Service Name when the dynamic target interface has no pattern variables.</p> <p>If the dynamic target interface has pattern variables, the Endpoint Mechanism field is set to Pattern Variables by default.</p>

Input Tab

When the **Endpoint Reference Mechanism** field on the **General** tab is set to Service & Namespace, the input elements for this task are serviceName and serviceNamespace. Any value you specify for these input elements overrides the value specified on the **Pattern Map** tab of the specified dynamic target interface.

When the **Endpoint Reference Mechanism** field on the **General** tab is set to Pattern Variables, then the pattern variables defined on the specified dynamic target interface are the input elements. This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the **Pattern Map** tab of the dynamic target interface.

Use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values.

The content of the **Input** tab depends on which Endpoint Reference Mechanism you select on the **General** tab — Application & Service Name, Application & Service URI, or Pattern Variables:

Application & Service Name	<p>When the Endpoint Reference Mechanism field on the General tab is set to Application & Service Name, the input elements for the Set Dynamic Reference task are <code>ApplicationName</code> and <code>ServiceName</code>:</p> <ul style="list-style-type: none"> • <i>ApplicationName</i> refers to the application name provided during deployment of a composite, to uniquely identify an instance of an application template. • <i>ServiceName</i> is the name of the composite service that is contained in the target composite.
Application & Service URI	<p>When the Endpoint Reference Mechanism field on the General tab is set to Application & Service URI, the input element for the Set Dynamic Reference task is <code>ApplicationServiceURI</code>.</p> <p>The data type of the ApplicationServiceURI input field is a URI of the format</p> <pre>urn:amx:EnvironmentName/ ApplicationName#service(ServiceName)</pre> <p>The variables <i>EnvironmentName</i>, <i>ApplicationName</i>, and <i>ServiceName</i> refer to the environment and service that are being invoked:</p> <ul style="list-style-type: none"> • <i>EnvironmentName</i> is the name of the ActiveMatrix environment that contains the target service. • <i>ApplicationName</i> refers to the application name that is provided during deployment of a composite, to uniquely identify an instance of an application template. • <i>ServiceName</i> is the name of the composite service that is contained in the target composite.
URI	<p>When the Endpoint Reference Mechanism field on the General tab is set to URI, the input elements for the Set Dynamic Reference task are URI.:</p> <ul style="list-style-type: none"> • SOAP over HTTP <pre>http://<HostName>:<PortNumber>/<Path></pre> <p><i><PortNumber></i> and <i><Path></i> are optional elements.</p> • SOAP over JMS: Specify the queue as <pre>jms:queue:<QueueName></pre> • ActiveMatrix Service Virtualization <pre>urn:amx:<EnvironmentName>/ <ApplicationName>#service(<PromotedServiceName>)</pre>

Pattern Variables

When the **Endpoint Reference Mechanism** field on the **General** tab is set to Pattern Variables, the pattern variables that are defined on the specified dynamic target interface are the input elements.

This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the **Pattern Map** tab of the dynamic target interface. You can use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values.

Set Exchange Variable



The Set Exchange Variable task sets the value of the exchange variable mediation exchange.

The Set Exchange Variable task can be placed on an input, output, or fault path.

The Set Exchange Variable task sets the value of the entire exchange variable — if you need to set several attributes, set them all at once, using one Set Exchange Variable task.

See [Working with Exchange Variables](#) for a description of how to define exchange variables for mediation operations.

General Tab

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Input Tab

The **Input** tab contains a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Transform Tasks for more information about using a mapping panel.

Field	Description
ExchangeVariable	<p>The exchange variable element defined in the mediation operation General tab appears on the right side of the mapper.</p> <p>To set the exchange variable, map values from the mediation exchange to the exchange variable element.</p>

Throw Fault



The Throw Fault task enables you to explicitly throw a fault in a mediation flow.

This task can be placed only on the input path. The Throw Fault task is useful in these situations:

- You want to deprecate a mediation operation, and send a fault to all clients that request that operation.
- You want to specify routing cases where a fault should be sent. For example, if a loan processing application cannot process loans over \$5,000,000, then you would configure a routing case for the loan request operation to examine the loan amount and place a Throw Fault task on the sub-path when the loan amount is over \$5,000,000.

See [Fault Processing in a Mediation Flow](#).

General Tab

- Use the **General** tab to select the fault to throw.
- You can choose to throw the `MediationTaskFault` message, or you can click **Browse** to open a dialog of service descriptors.
- You can choose from the list of messages in the selected service descriptors to send a specific message when a fault is thrown. When a message in a service descriptor is selected, the **WSDL Location** field appears.
- You can click the field label link to view the service descriptor in the WSDL editor.

Transform



The Transform task is used to manipulate the data available in a mediation flow so that the expected input, output, or fault message can be created.

Transform tasks can be placed on input, output, or fault paths. See [Transform Tasks](#).

General Tab

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Field	Description
Contribute Output to Mediation Exchange	<p>When not selected, this option signifies that the output of the Transform task should change the message data.</p> <p>When this option is selected, the message data is left unchanged, and the output of this task is added as another data item within the mediation exchange. The data contributed by this task is available to subsequent mediation tasks along the same path.</p> <p>If you select Use External Stylesheet on this tab, the Contribute Output to Mediation Exchange option is automatically selected and cannot be cleared.</p>
Use External Stylesheet	Select to use an external stylesheet for data transformation. This enables you specify the transformation mapping in your workspace, outside the mediation flow.
Input and Output Style	<p>Appears only if you select Use External Stylesheet.</p> <p>Specify how the XML will appear:</p> <ul style="list-style-type: none"> • Text Specified with a string. • Binary Specified with a binary value. • Tree Specified with an any element, so that you can transform data already in an XML document.
Stylesheet Reference Type	<p>Appears only if you select Use External Stylesheet.</p> <p>Select the type of reference for the Transform task:</p> <ul style="list-style-type: none"> • A static reference enables you to select a single (static) stylesheet from a folder that is in your project. • A dynamic reference enables you to select a set of stylesheets from a folder in the project. At run-time one of the stylesheets in the list will be used dynamically, based on the value provided for the <i>stylesheetURI</i> element that is in the Input tab of the mediation task. <p>For example, if the folder specified for the dynamic reference is <i>MySOAPProject/Service Descriptors</i> and the stylesheet is in the folder <i>MySOAPProject/Service Descriptors/folder1/sample.xml</i>, the value that must be provided for the <i>stylesheetURI</i> element must be <i>folder1/sample.xml</i>.</p> <p>When you specify a folder for dynamic reference, ActiveMatrix recursively includes the stylesheets under this folder and its sub-folders.</p>
Static Style Sheet Reference	Appears if you select a static stylesheet reference type. Click Browse to select a single (static) stylesheet that is in your workspace.
Dynamic Stylesheet Folder	Appears if you selected a dynamic stylesheet reference type. Chose a value available in the drop-down list. At run-time, one of the style sheets in the list will be used dynamically, based on the input to the mediation task.

Input Tab

Use the Input tab to map data from the mediation exchange into the expected message schema.

Field	Description
xmlString	Specify an XML document serialized as a string.
xmlBinary	Specify an XML document serialized in Base64Binary format.
xmlTree	Specify an XML document.
stylesheetURI	Specify the schema URI, so that ActiveMatrix can locate it at run time and use it for the transformation.
parameter	<p>A stylesheet can expect zero, one or more parameter(s) for its execution at runtime:</p> <ul style="list-style-type: none"> • Parameter Name — Name of the parameter the stylesheet expects. • Parameter Value — Value of the parameter.

Output Tab

The Output tab shows a tree representation of the Transform task output. Depending on the input style chosen, the output can be:

- xmlString — XML document serialized as string
- xmlBinary — XML document serialized in Base64Binary format
- xmlTree — XML document

Validate XML



The Validate XML task is used to validate message data, a WSDL message, XML text, binary, or XML tree formats against a schema.

The output of the Validate XML task is contributed to the mediation exchange, and can be used by downstream tasks. Validate XML processes an XML document against an XML schema, to report any errors found. It does not produce a parsed tree.

You choose the schema against which validation is to be performed by first specifying its reference type in the **General** tab of the Validate XML task:

- A *static* reference enables you to select a single (static) schema from a folder that is in your project.
- A *dynamic* reference enables you to select a set of schemas from a folder that is in your project. At run-time one of the schemas in the list will be used dynamically for validation, based on the input to the mediation task. When you specify a folder for dynamic reference, ActiveMatrix recursively includes the schemas under this folder and its sub-folders



The schema for a reference must be located in the same project as the mediation flow that uses it.

General Tab

On the **General** tab you specify a name and description for the task, and specify the type of schema to be used during verification.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Throw Fault on Validation Error	<p>When this option is set, an error in the Validate XML task results in a <code>MediationTaskFault</code>.</p> <p>When this check box is not selected, an error in the Validate XML task produces output that contains two fields:</p> <p>valid has two values: <i>true</i> if the XML is valid; <i>false</i> if the XML is invalid.</p> <p>error appears if the XML validation fails, and contains an <code>errorCode</code> and an <code>errorMessage</code>. These codes follow the W3C specification for XML schema.</p>
Input	<p>Select the input type:</p> <ul style="list-style-type: none"> • MessageData—Validates the mediation or target operation's input data. This option reads the input message itself, so no mapping is required. MessageData is the default input type. • WSDL Message—Validates the input or output of any operation in the WSDL. When you select the WSDL Message option, the Schema Source field opens. Select the mediation or target option, and the message type (input or output). • Text • Binary • XML Tree
Schema Reference Type	<p>When you select an input type of Text, Binary, or XML Tree, you can choose a static or dynamic reference type:</p> <ul style="list-style-type: none"> • A static reference enables you to select a single (static) schema from a folder that is in your project. • A dynamic reference enables you to select a set of schemas from a folder that is in your project. At run-time one of the schemas in the list will be used dynamically for validation, based on the input to the mediation task. When you specify a folder for dynamic reference, ActiveMatrix recursively includes the schemas under this folder and its sub-folders.

Field	Description
Schema Element or Schema Folder	<p>This field is based on whether you choose a static or dynamic reference type:</p> <ul style="list-style-type: none"> • If you choose Static Reference Type, the Schema Element field appears. Specify the XML schema document against which the incoming XML will be validated. • If you choose Dynamic Reference Type, the Schema Folder field appears. Specify the folder where schema resources are located.

Input Tab

The content of the Input tab depends on the Input type you selected on the **General** tab.

Field	Description
MessageData	No mapping is required in the Input tab.
WSDL Message	Displays a message tree corresponding to the operation and message selected in the General tab.
Text	<p>Specify the <code>xmlString</code> input to validate.</p> <p>If you chose a Dynamic Reference Type in the General tab, you can specify an optional <code>elementName</code>.</p>
Binary	<p>Specify the <code>xmlBinary</code> input to validate.</p> <p>If you chose a Dynamic Reference Type in the General tab, you can specify an optional <code>elementName</code>.</p>
Tree	<p>In the <code>xmlTree</code> node, specify any element to validate.</p> <p>If you chose a Dynamic Reference Type in the General tab, you can specify an optional <code>elementName</code>.</p>

Output Tab

The **Output** tab of the Validate XML task shows the results of the validation, indicating whether the incoming XML is valid or invalid, after being verified against the specified schema.

If validation fails, an error description identifies the cause of the failure. You can log this error description for design-time troubleshooting.

If the **Throw Fault** field is selected, no output is produced by this task. The **Output** tab shows a tree with the message `No Output Configured`.

XPath Route



The XPath Route task enables you to send messages to a specific destination based on conditions that you specify.

Data from the mediation exchange, such as the security information or message body, can be used to specify the conditions of the route.

XPath Route tasks can only be placed on input paths, but specifying an XPath Route task on the input path automatically creates the correct output and fault paths.

See [Routing Messages in a Mediation Flow](#) for more information about the XPath Route task.





General Tab

Use the **General** tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows.

Field	Description
Name	Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.
Description	Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.

Decision Tab

Use the **Decision** tab to create routing cases, routing conditions, and variables to hold data that will be evaluated in the routing conditions. The **Decision** tab includes a toolbar for adding and deleting cases and conditions

Toolbar Icon	Description
 Add Case	Adds a routing case to the table on this tab. A new case appears in the table on this tab with a default name, and the case is drawn in the mediation flow diagram. By default, new cases created with this icon point to an error icon until a Target Service/Operation is specified for the case.
 Delete Case	Deletes the selected routing case.
 Add Variable	<p>Adds a variable to use in routing conditions. Clicking this icon opens a dialog that enables you to specify the name and data type of the variable:</p> <ul style="list-style-type: none"> • String • Integer • Boolean • Float • Double • Decimal <p>Each variable appears as a column between the Case column and the Target Service/Operation column.</p>
 Delete Variable	Opens a dialog for you to select the variable to delete.

The **Decision** tab includes a table containing all of the routing cases.

Column	Description
Case	Name of the routing case. Click the cell to edit the name.
Variable Names (routing condition)	<p>The name of each variable that you create appears in at the top of the middle column of the table. The middle column is used to specify the XPath expression for the routing condition for each case. Your expressions are not limited to simple comparisons, and you do not need to use any of the variables you have defined in the expressions.</p> <p>You must type the XPath expression in the condition field next to each routing case, or select the field and use the XPath Editor field at the bottom of the tab to edit the expression.</p> <p>Variables are referenced in the XPath expressions for each routing case by their names. Unlike XPath expressions in the Transform task, you do not need to use a dollar sign to specify the root of the path to the variable. For example, the expression to determine if the city variable is equal to "Palo Alto" would be: <code>city = "Palo Alto"</code></p> <p>The Transform task has a graphical XPath editor that you can use as a reference for creating XPath functions for the route task. See Using XPath on page 107 and Data/Function Tabs on page 91 for more information about XPath.</p>
Target Service/ Operation	The name of the Target Service and target operation that is the destination for this case. If you drag the path for the case to a target operation in the mediation flow, this field is set automatically to the correct value. You can also click this field to either type or select the target operation.

At the bottom of the table is the configuration for the Otherwise case for the route. The Otherwise case is taken when all other cases evaluate to false. Use the **Target Service Operation** field to specify the target operation to perform for this case.

You can use the XPath editor window at the bottom of the **Decision** tab to edit the XPath expressions for each routing condition.

Input Tab

Use the **Input** tab to map data from the mediation exchange into the list of variables that you have created for the XPath Route task. See [Transform Tasks](#) for a description of how to use a mapping panel.

TIBCO AutoMediate Command-Line Tool

TIBCO AutoMediate Command Line is an independent command-line tool that enables you to quickly on-ramp a large number of existing services, without having to use TIBCO Business Studio, the Mediation Flow Editor, or the Composite Editor to build the necessary design-time components.

The AutoMediate Command Line tool uses existing services as input, specified in a concrete WSDL, to create a fully functional composite application with pass-through mediation capabilities. The tool generates a deployment artifact archive that you can deploy into the runtime environment as the first step in establishing an Enterprise Service Bus (ESB).

By establishing an ESB pattern using the AutoMediate Command Line tool and ActiveMatrix, you virtualize existing provided or consumed services so that they become location-transparent and more adaptable to change.

- Virtualizing provided services hides the location of service providers, helping to avoid interrupting clients that are using the services.
- Virtualizing consumed services hides the details of how the services are provided, helping to avoid interrupting logic that depends on the services.

The AutoMediate Command Line tool builds SOA projects that contain these components.

- A pass-through mediation flow.
- A composite that contains the mediation component wired to services and references, depending on the number of ports specified in the concrete WSDL.
- The deployment artifact file for the generated composite application.

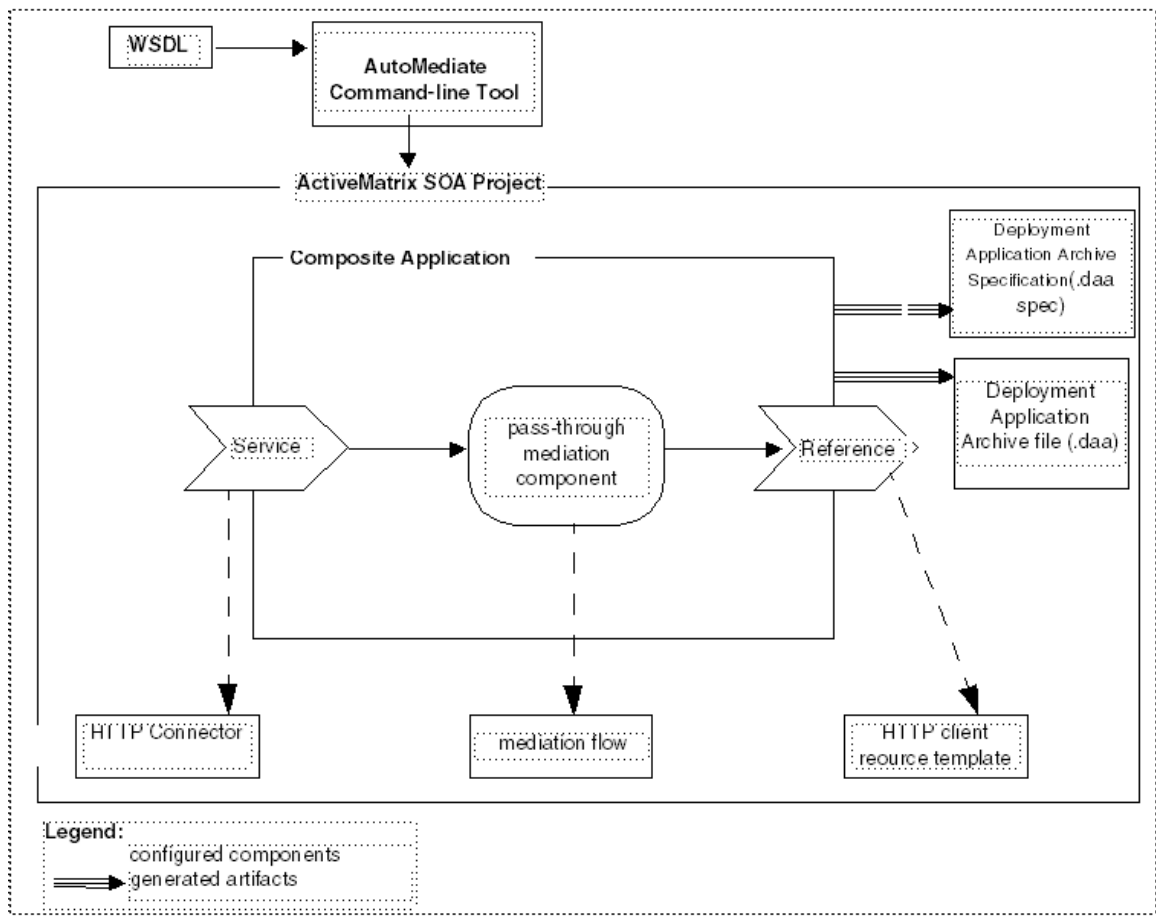
You can save these components in your source control system, then add mediation capabilities — for example routing, transformation, and validation — whenever changes are necessary.

AutoMediate Command-Line Tool Flow

The AutoMediate Command Line tool creates a SOA project from a concrete WSDL with HTTP service bindings.

In the following diagram:

- The Service is a SOAP service. The host can be configured during deployment using HTTP connector name generated for the service.
- The AutoMediate Command Line tool extracts binding information from the WSDL and uses it in the Reference. The tool also specifies the binding information as a substitution variable that can be replaced at run-time, if the Reference has changed its location.



Running the AutoMediate Command-Line Tool

The AutoMediate Command-Line tool creates a SOA project from a concrete WSDL with HTTP service bindings.

Procedure

1. The AutoMediate Command-Line tool is available in the `<TIBCO_HOME>/amx_it_mediation/<version>/bin` location. Navigate to this folder and open a command window.
2. At the prompt, type the AutoMediate command, specifying the concrete WSDL and any options. See [AutoMediate Command Syntax and Options](#).



The WSDL files you specify must be concrete WSDLs. This release of the AutoMediate Command-Line tool does not support abstract WSDLs.

The AutoMediate Command-Line tool executes, creating a SOA project from the concrete WSDL.

3. Import your SOA project into TIBCO Business Studio.

The new project includes the deployment application specification and the application archive file that contains the composite application.



After the AutoMediate Command Line creates your SOA project, you can deploy its deployment application archive (.daa file) directly into the runtime environment. See *TIBCO ActiveMatrix Administration Guide* for more information.

AutoMediate Command Syntax and Options

The **AutoMediate** command uses concrete WSDL file or directory locations as input and generates a composite application that provides pass-through mediation capabilities for existing web services.

Syntax

AutoMediate [-**projectName** *project_name*] [-**projectDir** *project_dir*] [-**serviceHost** *host_name*] [-**servicePort** *port*] *wsdl location*

AutoMediate Command Options

Option	Description
-projectName	<p>The name of the SOA project:</p> <ul style="list-style-type: none"> If you specify a name, a single SOA project is created. A single project is created even if you specify multiple WSDL files. If you do not specify a name, all the projects are generated based on the @name attribute of the WSDL definitions element. <p>If the @name attribute is not specified in the WSDL, the name of the WSDL file is used.</p>
-projectDir	<p>The directory where generated projects are to be created.</p> <p>If this option is not specified, generated projects are created in the current working directory.</p>
-serviceHost	<p>The host name for the service endpoint using SOAP over JMS.</p> <p>If this option is specified, the AutoMediate Command Line tool overwrites the host name field in the Naming Provider URL field in the generated resource template.</p>
-servicePort	<p>The port number for the service endpoint using SOAP over JMS.</p> <p>If this option is specified, the AutoMediate Command Line tool overwrites the port in the Naming Provider URL in the generated resource template.</p>
wsdl location	<p>Specify the location of the source WSDL files:</p> <ul style="list-style-type: none"> If you use a single WSDL file, specify the name of the source WSDL file or directory location. If you use multiple WSDL files, specify the name of the directory containing the source WSDL files. <p>If you do not specify a -projectName when you use multiple WSDL files, the AutoMediate Command Line tool creates one ActiveMatrix SOA project for each concrete WSDL file in the source directory.</p>

Option	Description
-httpConnectorName	<p>The name of the HTTP connector for services bound using SOAP over HTTP.</p> <p>If this option is not specified, HTTP connector name is the service name plus the port name. It is ignored for web services that are bound using SOAP over JMS.</p>
-serviceJmsConFactoryJndiName (optional)	<p>The JMS connection factory JNDI name for the resource template that is generated to configure a SOAP over JMS service endpoint.</p> <p>If this field is specified, the AutoMediate Command Line tool will overwrite the connection factory JNDI name in the generated JMS connection factory resource template for the service endpoint.</p>
-serviceJmsDestJndiName (optional)	<p>The JMS destination JNDI name for the resource template that is generated to configure a SOAP over JMS service endpoint.</p> <p>If this field is specified, the AutoMediate Command Line tool will overwrite the destination JNDI name in the generated JMS destination resource template for the service endpoint.</p>
-refJmsConFactoryJndiName (optional)	<p>The JMS connection factory JNDI name for the resource template that is generated to configure a SOAP over JMS reference endpoint.</p> <p>If this field is specified, the AutoMediate Command Line tool will overwrite the connection factory JNDI name in the generated JMS connection factory resource template for the reference endpoint.</p>
-refJmsDestJndiName (optional)	<p>The JMS destination JNDI name for the resource template that is generated to configure a SOAP over JMS reference endpoint.</p> <p>If this field is specified, the AutoMediate Command Line tool will overwrite the destination JNDI name in the generated JMS destination resource template for the reference endpoint.</p>
-daaOnly (optional)	Used to generate only the DAA file.

AutoMediate Command Examples

Example	Description
AutoMediate -projectName SOA webservice.wsdl	Automatically mediates one or more web services defined by single WSDL file.

Example	Description
AutoMediate -projectName SOA webService.wsdl webService1.wsdl webService2.wsdl AutoMediate c:/tibco/SOAP/wsdl <ul style="list-style-type: none"> c:/tibco/SOAP/wsdl1 c:/tibco/SOAP/wsdl2 AutoMediate webService.wsdl c:/tibco/ SOAP/wsdl webService1.wsdl	Automatically mediates web services.
AutoMediate -projectDir C:/tibco/SOA/ mediation/workspace webService.wsdl	Automatically mediates one or more web services and writes generated SOA project to a specific directory.
AutoMediate -projectDir C:/tibco/SOA/ mediation/workspace -serviceHost localhost -servicePort 9897 webService.wsdl	Automatically mediates one or more web services, generates a SOA project in a specific directory, and updates the host and port for the service.
AutoMediate -projectDir C:/tibco/SOA/ mediation/workspace c:/tibco/wsdl	Automatically mediates web services defined by the WSDL files contained in the specified folder.
AutoMediate webService.wsdl -daaOnly - projectDir C:/tibco/SOA/mediation/workspace	Automatically mediates one or more web services and generates the DAA in the specified directory.

AutoMediate Command-Line Exception

Exception	Description
WSDLFileNotFoundException	The WSDL location passed to the AutoMediate Command Line tool is invalid.
NoWSDLServiceDefinedException	<p>The WSDL passed to the AutoMediate Command Line tool does not have any services defined.</p> <p>AutoMediate Command Line supports only concrete WSDLs for this add-on pack release.</p>
NoWSDLServiceBindingException	<p>The port child element of a WSDL service element has a missing binding attribute.</p> <p>Binding must be provided for AutoMediate Command Line to generate a fully functional composite application.</p>

Exception	Description
NoWSDLBindingPortTypeException	A WSDL binding has a missing port type attribute. The port type must be provided for AutoMediate Command Line to generate a fully functional composite application.

AutoMediate ANT Command Syntax and Options

The **AutoMediate** ANT command generates a composite application that provides pass-through mediation capability for an existing web service or web-services. The WSDL URI format must be in EMF URI format.

Syntax

amx_eclipse_ant.exe [-DprojectDir=*project_dir*] [-DprojectName=*project_name*] [-DserviceHost *host_name*] [-DservicePort *port*] [-DwsdlLocation *wsdl location*] [-DwsdlLocations=*wsdl location*] [-buildfile *build_file*]

The AutoMediate ANT command is located in the <TIBCO_HOME>\studio\<version>\eclipse directory

See [AutoMediate Command and Syntax Options](#).

AutoMediate ANT Command-Line Examples

amx_eclipse_ant.exe -DprojectName=SOA webservice.wsdl -buildfile build.xml	Automatically mediates one or more web services defined by single WSDL file.
amx_eclipse_ant.exe -DprojectName= SOA - DwsdlLocation=webservice.wsdl;webservice 1.wsdl;webservice2.wsdl -buildfile build.xml amx_eclipse_ant.exe -DprojectName= SOA - DwsdlLocation=c:\tibco\SOAP\wsdl1;c: \tibco\SOAP\wsdl2;c:\tibco\SOAP\wsdl3 - buildfile build.xml	Automatically mediates web services.
amx_eclipse_ant.exe -DprojectDir= c:/ tibco/SOA/mediation/workspace - DwsdlLocation=webservice.wsdl -buildfile build.xml	Automatically mediates one or more web services and writes generated SOA project to a specific directory.
amx_eclipse_ant.exe -DprojectDir=C:/ tibco/SOA/mediation/workspace - serviceHost localhost -servicePort 9897 webservice.wsdl -buildfile build.xml	Automatically mediates one or more web services, generates a SOA project in a specific directory, and updates the host and port for the service.
amx_eclipse_ant.exe -daaOnly - DprojectDir= c:/tibco/SOA/mediation/ workspace -DwsdlLocation=webservice.wsdl -buildfile build.xml	Automatically mediates one or more web services and generates the DAA in the specified directory.

Introduction to gXML Applications

gXML is a way of writing XML code in the Java language. The code that you write to the gXML API can be run against any data model that supports the gXML bridge.

A Generic Java API for XQuery Data Model (XDM) and eXtensible Markup Language (XML) Processing, gXML also provides a cohesive suite of XML processing implementations such as XPath, XSLT, XQuery, Serialization, W3C XML Schema and Validation.

This flexibility offers several benefits:

- Minimizes expensive conversion overhead.
- Increases opportunities for performance optimization.
- Increases code reuse.
- Minimizes risks associated with locking into one Data Model.

gXML currently supports Parsing, Serialization, XDM Data Model, XPath 2, XSLT 2 and XQuery, W3C XML Schema and Validation.

- A gXML bridge is provided for `org.w3c.dom.Node`.
- A gXML bridge for a high performance proprietary implementation is complete but not yet released.
- A gXML bridge for a reference implementation is complete but not yet released. A gXML bridge for AxiOM is in the works.

Developing gXML Applications

All gXML processors, including custom processing, run within a `GxProcessingContext` instance that provides necessary metadata. A `GxProcessingContext` instance in turn is created through a `GxApplication` instance.

You must write a class that provides an instance of `GxApplication`. The best way to do this is to write an abstract class that implements all but the `newProcessingContext` method of `GxApplication`. This approach allows you to write your application generically and then inject the choice of parameterization as late as possible for maximum code reuse and flexibility.

This, of course, is not the only way to use gXML. An existing architecture may force the choice of parameterization and create silos of XML processing. The degree of integration in this case may be less than is possible with a homogeneous solution.

Whatever the approach, the best way to use gXML is to write generic, parameterized, and XML processing code whenever possible.

Implementing GxApplication

You must write a class that provides an instance of `GxApplication`. The best way to do this is to write an abstract class that implements all but the `newProcessingContext` method of `GxApplication`.

```
001 package org.gxml.book.common;
002
003 import java.io.StringWriter;
004 import java.net.URI;
005 import java.net.URISyntaxException;
006
007 import junit.framework.TestCase;
008
009 import org.gxml.sa.GxApplication;
010 import org.gxml.sa.GxModel;
011 import org.gxml.sa.GxNameBridge;
012 import org.gxml.sa.GxProcessingContext;
```

```

013 import org.gxml.sa.GxSequenceHandler;
014 import org.gxml.xdm.Resolver;
015
016 import com.tibco.gxml.sa.api.common.util.PreCondition;
017 import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
018 import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
019
020 public abstract class SampleApp<I, U, N extends I, A extends I, S, T, X>
021 extends TestCase implements GxApplication<I, U, N, A, S, T, X>
022 {
023     public Resolver getResolver()
024     {
025         try
026         {
027             return new SampleResolver(new URI("../plugins/org.gxml.book/
resources/foo.xml"));
028         }
029         catch (final URISyntaxException e)
030         {
031             throw new AssertionError(e);
032         }
033     }
034
035     protected String serialize(final N node, final GxProcessingContext<I, U, N,
A, S, T, X> pcx)
036     {
037         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
038
039         // Configure for "pretty" printing.
040         sf.setIndent(Boolean.TRUE);
041
042         final StringWriter w = new StringWriter();
043
044         final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
045
046         final GxModel<N, A, S, T> model = pcx.getModel();
047
048         handler.startDocument(null);
049         try
050         {
051             model.stream(node, true, true, handler);
052         }
053         finally
054         {
055             handler.endDocument();
056         }
057
058         return w.toString();
059     }
060
061     /**
062     * Some bridge implementations may use {@link String} directly for symbols.
063     They must make them behave according to
064     * symbol semantics (==, toString).
065     */
066     public void assertNodeSymbolSemantics(final N node, final
GxProcessingContext<I, U, N, A, S, T, X> pcx)
067     {
068         final GxModel<N, A, S, T> model = pcx.getModel();
069         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
070
071         switch (model.getNodeKind(node))
072         {
073             case ELEMENT:
074             {
075                 assertSymbolSemantics(model.getNamespaceURI(node), nameBridge);
076                 assertSymbolSemantics(model.getLocalName(node), nameBridge);
077             }
078             case TEXT:
079             case DOCUMENT:
080             {

```

```

079
080     }
081     break;
082     default:
083     {
084         throw new AssertionError(model.getNodeKind(node));
085     }
086 }
087 }
088
089 public void assertSymbolSemantics(final S symbol, final GxNameBridge<S>
nameBridge)
090 {
091     PreCondition.assertArgumentNotNull(symbol, "symbol");
092     PreCondition.assertArgumentNotNull(nameBridge, "nameBridge");
093     assertSame(symbol, nameBridge.symbolize(symbol.toString()));
094     assertSame(symbol, nameBridge.symbolize(copy(symbol.toString())));
095 }
096
097 /**
098  * Do anything to manufacture a String that is equal, but not identical
(the same), as the original.
099  * <p>
100  * This method has the post-condition that the strings are equal but not
the same.
101  * </p>
102  *
103  * @param original
104  *       The original.
105  * @return A copy of the original string.
106  */
107 private String copy(final String original)
108 {
109     final String copy = original.concat("junk").substring(0,
original.length());
110     // Post-conditions verify that this actually works and isn't
"optimized" out.'
111     assertEquals(original, copy);
112     assertNotSame(original, copy);
113     // Be Paranoid
114     assertTrue(original.equals(copy));
115     assertFalse(original == copy);
116     // OK. That'll do.'
117     return copy;
118 }
119 }

```

Implementing GxCatalog

A catalog provides the means to isolate your application from the physical location of file resources. Writing a catalog means implementing the GxCatalog interface so that it maps from the logical locations specified in code or XML resources to the corresponding physical location.

```

001 package org.gxml.book.common;
002
003 public class SampleCatalog
004 {
005
006 }

```

Implementing GxResolver

A resolver takes a base-uri and an href and uses these two values to return a stream.

```

001 package org.gxml.book.common;
002
003 import java.io.File;
004 import java.io.FileNotFoundException;
005 import java.io.IOException;
006 import java.io.InputStream;
007 import java.net.URI;

```

```

008 import java.net.URISyntaxException;
009 import java.net.URL;
010
011 import org.gxml.xdm.Resolved;
012 import org.gxml.xdm.Resolver;
013
014 import com.tibco.gxml.sa.api.common.util.PreCondition;
015
016 public final class SampleResolver implements Resolver
017 {
018     final URI baseURI;
019
020     public SampleResolver(final URI baseURI)
021     {
022         this.baseURI = PreCondition.assertArgumentNotNull(baseURI, "baseURI");
023     }
024
025     /**
026      * Convert a URI relative to a base URI into an input source.
027      * <p/>
028      * This default implementation requires that neither parameter be null, and
029      * performs the expected action to retrieve
030      * the input source (which may involve network access).
031      *
032      * @param baseURI
033      *         the base URI against which the target is to be resolved; must
034      *         not be null
035      * @param location
036      *         the URI to resolve; must not be null
037      * @return a pair of InputStream and resolved URI.
038      */
039     public Resolved<InputStream> resolveInputStream(final URI location) throws
040     IOException
041     {
042         PreCondition.assertArgumentNotNull(location, "uri");
043         if (location.isAbsolute())
044         {
045             return retrieve(location, location);
046         }
047         else
048         {
049             PreCondition.assertArgumentNotNull(baseURI, "baseURI");
050
051             final URI base = baseURI.normalize();
052             final URI resolved = base.resolve(location);
053
054             return retrieve(location, resolved);
055         }
056     }
057
058     private Resolved<InputStream> retrieve(final URI location, final URI uri)
059     throws IOException
060     {
061         PreCondition.assertArgumentNotNull(uri, "uri");
062
063         final URL toRetrieve;
064
065         if (!uri.isAbsolute()) // assume local file
066         {
067             final File canonFile = new File(uri.toString()).getCanonicalFile();
068             toRetrieve = canonFile.toURI().toURL();
069         }
070         else
071         {
072             toRetrieve = uri.toURL();
073         }
074
075         if (toRetrieve == null)
076         {
077             throw new FileNotFoundException(uri.toString());
078         }
079     }
080 }

```

```

076         final InputStream stream = toRetrieve.openStream();
077         if (stream == null)
078         {
079             throw new FileNotFoundException(toRetrieve.toString());
080         }
081         try
082         {
083             return new Resolved<InputStream>(location, stream,
toRetrieve.toURI());
084         }
085         catch (final URISyntaxException e)
086         {
087             throw new AssertionError(e);
088         }
089     }
090 }

```

Injecting DOM

The final task in providing a concrete GxApplication class is to implement the `newProcessingContext` method on a derived class. You choose the tree, atomic values, metadata and symbols that your application will use. In many cases you can use an off-the-shelf processing context class, but you can also assemble or build your own.

If you are going to use gXML with `org.w3c.dom.Node`, you have choices for the atomic values that your system will use as well as the metadata implementation. This example uses atomic values that are mostly Java wrapper types and the reference sequence type implementation, `SmSequenceType`.

```

001 package org.gxml.book.parsing;
002
003 import org.gxml.sa.GxMetaBridge;
004 import org.gxml.sa.GxNameBridge;
005 import org.gxml.sa.mutable.GxApplicationMutable;
006 import org.gxml.sa.mutable.GxProcessingContextMutable;
007 import org.gxml.xs.SmMetaBridge;
008 import org.gxml.xs.SmSequenceType;
009 import org.w3c.dom.Node;
010
011 import com.tibco.gxml.sa.api.common.datatype.StringNameBridge;
012 import com.tibco.gxml.sa.common.atom.AtomBridge;
013 import com.tibco.gxml.sa.common.helpers.GxMetaBridgeOnSmMetaBridgeAdapter;
014 import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
015 import com.tibco.gxml.sa.xdm.dom.DomProcessingContext;
016 import com.tibco.gxml.xs.SmMetaBridgeFactory;
017
018 /**
019  * Demonstration of constructing a concrete GxApplication(Mutable)
implementation
using the DOM processing context.
020  */
021 public final class DomValidatingParsingSample extends
BookValidatingParsingSample<Object, Object, Node, Object, String,
SmSequenceType<Object,
String>, Object> implements GxApplicationMutable<Object, Object, Node, Object,
String,
SmSequenceType<Object, String>, Object>
022 {
023     public final GxProcessingContextMutable<Object, Object, Node, Object,
String,
SmSequenceType<Object, String>, Object> newProcessingContext()
024     {
025         // The name bridge is created along with the processing context for
maximum
concurrency.
026         final GxNameBridge<String> nameBridge = new StringNameBridge();
027         final AtomBridge<String> atomBridge = new
AtomBridge<String>(nameBridge);
028         final SmMetaBridge<Object, String> cache = new
SmMetaBridgeFactory<Object,
String>(new SmAtomBridgeOnGxAtomBridgeAdapter<Object,

```

```

String>(atomBridge)).newMetaBridge();
029         final GxMetaBridge<Object, String, SmSequenceType<Object, String>>
metaBridge = new GxMetaBridgeOnSmMetaBridgeAdapter<Object, String>(cache,
atomBridge);
030
031         final DomProcessingContext<Object, SmSequenceType<Object, String>>
pcx = new DomProcessingContext<Object, SmSequenceType<Object, String>>
(this, metaBridge, cache);
032
033         // Set the "owning" processing context on the atom bridge.
034         atomBridge.setProcessingContext(pcx);
035
036         // Return the newly constructed processing context.
037         return pcx;
038     }
039 }

```

gXML Recipes

Parsing a Character Stream and a Byte Stream

```

001 package org.gxml.book.parsing;
002
003 import java.io.InputStream;
004 import java.io.Reader;
005 import java.io.StringReader;
006 import java.net.URI;
007
008 import org.gxml.book.common.SampleApp;
009 import org.gxml.sa.GxModel;
010 import org.gxml.sa.GxNameBridge;
011 import org.gxml.sa.GxProcessingContext;
012 import org.gxml.xdm.NodeKind;
013 import org.gxml.xdm.Resolved;
014 import org.gxml.xdm.Resolver;
015
016 import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
017 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
018 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
019
020 public abstract class BookIntroParsingSample<I, U, N extends I, A extends I, S,
T,
X> extends SampleApp<I, U, N, A, S, T, X>
021 {
022     public void testCharacterStreamParse() throws Exception
023     {
024         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
025
026         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N,
A, S, T, X>(pcx);
027
028         final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
029
030         final String xmlString = "<e>123</e>";
031         final URI systemId = new URI("e.xml");
032         final Reader characterStream = new StringReader(xmlString);
033         final N doc = builder.parse(characterStream, systemId);
034
035         final GxModel<N, A, S, T> model = pcx.getModel();
036
037         assertEquals(NodeKind.DOCUMENT, model.getNodeKind(doc));
038
039         final N e = model.getFirstChildElement(doc);
040         assertEquals(NodeKind.ELEMENT, model.getNodeKind(e));

```

```

041         assertEquals("e", model.getLocalNameAsString(e));
042         assertEquals("123", model.getStringValue(e));
043     }
044
045     public void testByteStreamParse() throws Exception
046     {
047         final Resolver resolver = getResolver();
048
049         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
050
051         final URI systemId = new URI("email.xml");
052         final Resolved<InputStream> source =
resolver.resolveInputStream(systemId);
053
054         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N,
A, S, T, X>(pcx);
055
056         final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
057
058         final N document = builder.parse(source.getResource(),
source.getSystemId());
059
060         final GxModel<N, A, S, T> model = pcx.getModel();
061
062         assertEquals(NodeKind.DOCUMENT, model.getNodeKind(document));
063
064         final N email = model.getFirstChildElement(document);
065         assertEquals(NodeKind.ELEMENT, model.getNodeKind(email));
066         assertEquals("email", model.getLocalNameAsString(email));
067         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
068         final S namespaceURI = nameBridge.symbolize("http://www.example.com");
069         final S localName = nameBridge.symbolize("from");
070         final N from = model.getFirstChildElementByName(email, namespaceURI,
localName);
071         assertEquals("Julie", model.getStringValue(from));
072
073         for (final N node : model.getDescendantOrSelfAxis(document))
074         {
075             assertNodeSymbolSemantics(node, pcx);
076         }
077     }
078 }

```

Constructing a Data Model Tree Programmatically

This example demonstrates constructing a tree directly using the fragment builder.

```

001 package org.gxml.book.snoopy;
002
003 import java.io.IOException;
004 import java.io.InputStream;
005 import java.io.StringReader;
006 import java.io.StringWriter;
007 import java.net.URI;
008 import java.net.URISyntaxException;
009
010 import javax.xml.namespace.QName;
011 import javax.xml.parsers.ParserConfigurationException;
012
013 import org.gxml.book.common.SampleApp;
014 import org.gxml.sa.GxException;
015 import org.gxml.sa.GxFragmentBuilder;
016 import org.gxml.sa.GxMetaBridge;
017 import org.gxml.sa.GxModel;
018 import org.gxml.sa.GxNameBridge;
019 import org.gxml.sa.GxProcessingContext;
020 import org.gxml.sa.GxSequenceHandler;
021 import org.gxml.sa.GxVariantBridge;
022 import org.gxml.xdm.NodeKind;
023 import org.gxml.xdm.Resolved;

```

```

024 import org.gxml.xdm.Resolver;
025 import org.gxml.xs.SmName;
026
027 import com.tibco.gxml.sa.api.common.lang.ExprException;
028 import com.tibco.gxml.sa.api.common.lang.ExprResult;
029 import com.tibco.gxml.sa.api.common.lang.GxExpr;
030 import com.tibco.gxml.sa.api.common.lang.GxExprContextDynamicArgs;
031 import com.tibco.gxml.sa.api.common.lang.GxExprContextStaticArgs;
032 import com.tibco.gxml.sa.api.common.lang.GxLanguageToolKit;
033 import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
034 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
035 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
036 import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
037 import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
038 import com.tibco.gxml.sa.processor.xquery.LanguageToolKit;
039 import com.tibco.gxml.sa.processor.xslt.GxTransform;
040 import com.tibco.gxml.sa.processor.xslt.GxTransformBuilder;
041 import com.tibco.gxml.sa.processor.xslt.GxTransformer;
042 import com.tibco.gxml.sa.processor.xslt.XSLTransformBuilder;
043 import com.tibco.gxmlsa.processor.org.exslt.strings.ExsltStringsFunctionGroup;
044
045 public abstract class SnoopySample<I, U, N extends I, A extends I, S, T, X>
    extends
    SampleApp<I, U, N, A, S, T, X>
046 {
047     public void testDocumentFromString()
048     {
049         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
050
051         final N document = documentFromString(pcx);
052
053         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
054
055         sf.setIndent(true);
056
057         final StringWriter sw = new StringWriter();
058
059         final GxSequenceHandler<A, S, T> serializer = sf.newSerializer(sw);
060
061         final GxModel<N, A, S, T> model = pcx.getModel();
062
063         model.stream(document, true, true, serializer);
064
065         // System.out.println(sw.toString());
066     }
067
068     public void testFragmentBuilder()
069     {
070         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
071
072         final N document = documentFromEvents(pcx);
073
074         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
075
076         sf.setIndent(true);
077
078         final StringWriter sw = new StringWriter();
079
080         final GxSequenceHandler<A, S, T> serializer = sf.newSerializer(sw);
081
082         final GxModel<N, A, S, T> model = pcx.getModel();
083
084         model.stream(document, true, true, serializer);
085
086         // System.out.println(sw.toString());
087     }
088
089     private N documentFromString(final GxProcessingContext<I, U, N, A, S, T, X>

```

```

pcx)
090     {
091         final String strval = "" + "<?xml version='1.0' encoding='UTF-8'?>" +
"<book isbn='0836217462'>" + "
<title>Being a Dog Is a Full-Time Job</title>" + " <author>Charles M. Schultz</
author>" + " <character>" + "
<name>Snoopy</name>" + " <since>1950-10-04</since>" + " </character>" + "</book>";
092
093         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
094
095         final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
096
097         try
098         {
099             return builder.parse(new StringReader(strval), null);
100         }
101         catch (final IOException e)
102         {
103             throw new AssertionError();
104         }
105     }
106
107     private N documentFromEvents(final GxProcessingContext<I, U, N, A, S, T, X>
pcx)
108     {
109         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
110
111         final S NULL_NS_URI = nameBridge.empty();
112         final S BOOK = nameBridge.symbolize("book");
113         final S ISBN = nameBridge.symbolize("isbn");
114         final S TITLE = nameBridge.symbolize("title");
115         final S AUTHOR = nameBridge.symbolize("author");
116         final S CHARACTER = nameBridge.symbolize("character");
117         final S NAME = nameBridge.symbolize("name");
118         final S SINCE = nameBridge.symbolize("since");
119
120         final GxFragmentBuilder<N, A, S, T> builder = pcx.newFragmentBuilder();
121
122         // Note: Using try...finally not only ensures that elements get closed
when errors
123         // occur, it also helps to remind you to end elements and makes the
levels in
124         // the XML more obvious.
125         builder.startDocument(null);
126         try
127         {
128             builder.startElement(NULL_NS_URI, BOOK, "", null);
129             try
130             {
131                 builder.attribute(NULL_NS_URI, ISBN, "", "0836217462");
132                 builder.startElement(NULL_NS_URI, TITLE, "", null);
133                 try
134                 {
135                     builder.text("Being a Dog Is a Full-Time Job");
136                 }
137                 finally
138                 {
139                     builder.endElement();
140                 }
141                 builder.startElement(NULL_NS_URI, AUTHOR, "", null);
142                 try
143                 {
144                     builder.text("Charles M. Schultz");
145                 }
146                 finally
147                 {
148                     builder.endElement();
149                 }
150                 builder.startElement(NULL_NS_URI, CHARACTER, "", null);
151                 try
152                 {

```

```

153         builder.startElement(NULL_NS_URI, NAME, "", null);
154         try
155         {
156             builder.text("Snoopy");
157         }
158         finally
159         {
160             builder.endElement();
161         }
162         builder.startElement(NULL_NS_URI, SINCE, "", null);
163         try
164         {
165             builder.text("1950-10-04");
166         }
167         finally
168         {
169             builder.endElement();
170         }
171     }
172     finally
173     {
174         builder.endElement();
175     }
176 }
177 finally
178 {
179     builder.endElement();
180 }
181 }
182 finally
183 {
184     builder.endDocument();
185 }
186
187 return builder.getNodes().get(0);
188 }
189
190 public void testExample() throws ParserConfigurationException, IOException,
191 GxException, ExprException,
192 URISyntaxException
193 {
194     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
195 newProcessingContext();
196
197     final Resolver resolver = getResolver();
198
199     final URI xmlSystemId = new URI("hotel.xml");
200     final Resolved<InputStream> xmlInput =
201 resolver.resolveInputStream(xmlSystemId);
202
203     final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I,
204 U, N, A, S, T, X>(pcx);
205
206     final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
207
208     final N document = builder.parse(xmlInput.getResource(),
209 xmlInput.getSystemId());
210
211     final URI xslSystemId = new URI("hotel.xsl");
212     final Resolved<InputStream> xslInput =
213 resolver.resolveInputStream(xslSystemId);
214
215     final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
216 XSLTransformBuilder<I, U, N, A, S, T, X>
217 (pcx);
218
219     // poem.xsl uses version="2.0", but we want to use XPath 1.0
220 compatibility mode
221     // so that arguments to functions are converted etc.
222     compiler.setCompatibleMode(true);
223
224     final GxTransform<I, U, N, A, S, T, X> compiled =

```

```

compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());
215
216     final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
217
218     // TODO: Extract the configuration?
219     // compiled.configure(sf);
220
221     sf.setIndent(true);
222
223     final StringWriter w = new StringWriter();
224
225     final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
226
227     final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
228
229     transformer.transform(document, pcx, handler);
230 }
231
232 public void testVariableBinding() throws ParserConfigurationException,
IOException, GxException,
ExprException, URISyntaxException
233 {
234     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
235
236     final Resolver resolver = getResolver();
237
238     final URI xslSystemId = new URI("email.xsl");
239     final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);
240
241     final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T,
X>(pcx);
242
243     final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());
244
245     final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
246
247     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
248     final SmName<S> varName = nameBridge.name(new QName("to"));
249     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
250     final X value = valueBridge.stringValue("David");
251
252     transformer.bindVariableValue(varName, value);
253     transformer.bindVariableValue(nameBridge.name(new QName("http://
www.example.com", "from")),
valueBridge.stringValue("Julie"));
254
255     final N documentNode = transformer.transform(null, pcx);
256
257     final GxModel<N, A, S, T> model = pcx.getModel();
258
259     assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
260     final N email = model.getFirstChildElement(documentNode);
261     final N to = model.getFirstChildElementByName(email,
nameBridge.symbolize("http://www.example.com"),
nameBridge.symbolize("to"));
262     assertEquals("David", model.getStringValue(to));
263     final N from = model.getFirstChildElementByName(email, null,
nameBridge.symbolize("from"));
264     assertEquals("Julie", model.getStringValue(from));
265     final N again = model.getFirstChildElementByName(email,
nameBridge.symbolize("http://www.example.com"),
null);
266     assertEquals("David", model.getStringValue(again));

```

```

267     }
268
269     public void testExternalFunctions() throws ParserConfigurationException,
IOException, GxException,
ExprException, URISyntaxException
270     {
271         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
272
273         final Resolver resolver = getResolver();
274
275         final URI xmlSystemId = new URI("exslt.xml");
276         final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);
277
278         final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I,
U, N, A, S, T, X>(pcx);
279
280         final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
281
282         final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
283
284         final URI xslSystemId = new URI("exslt.xsl");
285         final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);
286
287         final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>
(pcx);
288
289         final String namespaceURI = "http://exslt.org/strings";
290         final ExsltStringsFunctionGroup<I, U, N, A, S, T, X> functions =
new ExsltStringsFunctionGroup<I, U, N, A, S, T, X>(namespaceURI, pcx);
291         compiler.setFunctionSigns(namespaceURI, functions);
292         compiler.setFunctionImpls(namespaceURI, functions);
293
294         final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());
295
296         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
297
298         // TODO: Extract the configuration.
299         // compiled.configure(sf);
300
301         sf.setIndent(true);
302
303         final StringWriter w = new StringWriter();
304
305         final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
306
307         final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
308
309         transformer.transform(document, pcx, handler);
310
311         // System.out.println(w.toString());
312     }
313
314     public void testHotel() throws ParserConfigurationException, IOException,
GxException, ExprException,
URISyntaxException
315     {
316         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
317
318         final Resolver resolver = getResolver();
319
320         final URI xmlSystemId = new URI("hotel.xml");
321         final Resolved<InputStream> xmlInput =

```

```

resolver.resolveInputStream(xmlSystemId);
322
323     final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I,
U, N, A, S, T, X>(pcx);
324
325     final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
326
327     final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
328
329     final URI xslSystemId = new URI("hotel.xsl");
330     final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);
331
332     final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>
(pcx);
333
334     final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());
335
336     final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
337
338     // TODO: Extract the configuration.
339     // compiled.configure(sf);
340
341     sf.setIndent(true);
342
343     final StringWriter w = new StringWriter();
344
345     final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
346
347     final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
348     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
349     final SmName<S> varName = nameBridge.name(new QName("MessageData"));
350     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
351     final X value = valueBridge.node(document);
352
353     transformer.bindVariableValue(varName, value);
354
355     transformer.transform(null, pcx, handler);
356
357     // System.out.println(w.toString());
358 }
359
360 public void testHelloWorld() throws Exception
361 {
362     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
363     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
364
365     final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new
LanguageToolKit<I, U,
N, A, S, T, X>(pcx);
366
367     final GxExprContextStaticArgs<I, U, N, A, S, T, X> senv =
xtk.newStaticContextArgs();
368     final String NAMESPACE = "http://www.peanuts.com";
369
370     senv.getInScopeNamespaces().declarePrefix("nuts",
nameBridge.symbolize(NAMESPACE));
371
372     final SnoopyFunctionGroup<I, U, N, A, S, T, X> peanutsFunctionGroup =
new
SnoopyFunctionGroup<I, U, N, A, S, T, X>(NAMESPACE, pcx);
373     senv.setFunctionSigns(NAMESPACE, peanutsFunctionGroup);
374     senv.setFunctionImpls(NAMESPACE, peanutsFunctionGroup);
375
376     final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();

```

```

377
378         final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare
("nuts:GetVariableProperty('foo','bar')", metaBridge.emptyType(), senv);
379
380         final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
381
382         final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg =
xtk.newDynamicContextArgs();
383
384         final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);
385
386         assertEquals("Bingo!", strval);
387     }
388 }

```

Validating

```

001 package org.gxml.book.parsing;
002
003 import java.io.InputStream;
004 import java.net.URI;
005
006 import javax.xml.namespace.QName;
007
008 import org.gxml.book.common.SampleApp;
009 import org.gxml.sa.GxApplication;
010 import org.gxml.sa.GxAtomBridge;
011 import org.gxml.sa.GxModel;
012 import org.gxml.sa.GxNameBridge;
013 import org.gxml.sa.GxProcessingContext;
014 import org.gxml.xdm.Resolved;
015 import org.gxml.xdm.Resolver;
016 import org.gxml.xs.SmComponentBag;
017 import org.gxml.xs.SmExceptionCatcher;
018 import org.gxml.xs.SmMetaLoadArgs;
019 import org.gxml.xs.SmName;
020
021 import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
022 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
023 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
024 import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
025 import com.tibco.gxml.xs.W3cXmlSchemaParser;
026
027 public abstract class BookValidatingParsingSample<I, U, N extends I, A extends
I, S, T, X>
extends SampleApp<I, U, N, A, S, T, X>
028 {
029     public void testValidatingParse() throws Exception
030     {
031         final GxApplication<I, U, N, A, S, T, X> app = this;
032
033         final Resolver resolver = app.getResolver();
034
035         final SmMetaLoadArgs args = new SmMetaLoadArgs();
036
037         final SmExceptionCatcher errors = new SmExceptionCatcher();
038
039         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
app.newProcessingContext();
040
041         final Resolved<InputStream> resource =
getResolver().resolveInputStream(new URI
("email.xsd"));
042
043         final W3cXmlSchemaParser<A, S> parser = new W3cXmlSchemaParser<A, S>
(new SmAtomBridgeOnGxAtomBridgeAdapter<A, S>(pcx.getAtomBridge()));
044
045         final SmComponentBag<A, S> components =
parser.parse(resource.getLocation(),
resource.getResource(), resource.getId(), errors, args, pcx);

```

```

046
047     pcx.register(components);
048
049     pcx.lock();
050
051     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
052
053     assertEquals(0, errors.size());
054
055     final URI xmlURI = new URI("email.xml");
056     final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlURI);
057
058     final GxDocumentBuilderFactory<N, S> factory =
new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
059
060     // Enable validation of the XML input.
061     factory.setValidating(true, nameBridge.name(new QName("http://
www.example.com",
"email"))));
062
063     final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
064
065     // TODO: Need to catch errors...
066     // builder.setExceptionHandler(errors);
067
068     final N doc = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
069
070     assertEquals(0, errors.size());
071
072     // System.out.println(serialize(doc, pcx));
073
074     final GxModel<N, A, S, T> model = pcx.getModel();
075     final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();
076
077     final N email = model.getFirstChildElement(doc);
078     final S namespaceURI = nameBridge.symbolize("http://www.example.com");
079     final N sent = model.getFirstChildElementByName(email, namespaceURI,
nameBridge.symbolize("sent"));
080     assertNotNull("model.getFirstChildElementByName", sent);
081     final SmName<S> typeName = model.getTypeName(sent);
082     assertNotNull("model.getTypeName", typeName);
083     assertEquals("dateTime", typeName.toQName().getLocalPart());
084     final A dateTime = model.getTypedValue(sent).get(0);
085
086     //
assertTrue(metaBridge.sameAs(metaBridge.handle(pcx.getTypeDefinition(type)),
087         // metaBridge.getType(SmNativeType.DATETIME)));
088
089     assertEquals("2008-03-23T14:49:30-05:00",
atomBridge.getC14NForm(dateTime));
090 }
091 }

```

Navigation

```

001 package org.gxml.book.parsing;
002
003 import java.io.InputStream;
004 import java.net.URI;
005
006 import org.gxml.book.common.SampleApp;
007 import org.gxml.sa.GxModel;
008 import org.gxml.sa.GxNameBridge;
009 import org.gxml.sa.GxProcessingContext;
010 import org.gxml.xdm.Resolved;
011 import org.gxml.xdm.Resolver;
012
013 import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;

```

```

014 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
015 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
016
017 public abstract class BookNavigationParsingSample<I, U, N extends I, A extends
I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
018 {
019     public void testBooksByNealStephenson() throws Exception
020     {
021         final Resolver resolver = getResolver();
022
023         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
024
025         final URI systemId = new URI("books.xml");
026         final Resolved<InputStream> source =
resolver.resolveInputStream(systemId);
027
028         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
029
030         final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
031
032         final N doc = builder.parse(source.getResource(),
source.getSystemId());
033
034         final GxModel<N, A, S, T> model = pcx.getModel();
035
036         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
037
038         final S namespaceURI = nameBridge.symbolize("http://www.example.com/
books");
039
040         final N inventory = model.getFirstChildElementByName(doc, namespaceURI,
nameBridge.symbolize("inventory"));
041
042         for (final N book : model.getChildElementsByName(inventory,
namespaceURI, nameBridge.symbolize("book")))
043         {
044             boolean found = false;
045
046             for (final N author : model.getChildElementsByName(book,
namespaceURI, nameBridge.symbolize("author")))
047             {
048                 if (model.getStringValue(author).equals("Neal Stephenson"))
049                 {
050                     found = true;
051                     break;
052                 }
053             }
054
055             if (found)
056             {
057                 final N title = model.getFirstChildElementByName(book,
namespaceURI, nameBridge.symbolize("title"));
058
059                 System.out.println(model.getStringValue(title));
060             }
061         }
062     }
063
064     public void testPurchaseOrder() throws Exception
065     {
066         final Resolver resolver = getResolver();
067
068         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
069
070         final GxModel<N, A, S, T> model = pcx.getModel();
071         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
072
073         final URI systemId = new URI("PurchaseOrder.xml");
074         final Resolved<InputStream> source =
resolver.resolveInputStream(systemId);

```

```

074
075     final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
076
077     final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
078
079     final N po = builder.parse(source.getResource(), source.getSystemId());
080
081     final N root = model.getFirstChildElement(po);
082
083     final N items = model.getFirstChildElementByName(root, null,
nameBridge.symbolize("items"));
084
085     double total = 0;
086     for (final N item : model.getChildElementsByName(items, null,
nameBridge.symbolize("item")))
087     {
088         System.out.println("partNum:" + model.getAttributeStringValue(item,
nameBridge.empty(), nameBridge.symbolize("partNum")));
089
090         final N price = model.getFirstChildElementByName(item, null,
nameBridge.symbolize("USPrice"));
091         total += Double.valueOf(model.getStringValue(price)).doubleValue();
092     }
093     System.out.println("Grand total = " + total);
094 }
095 }

```

Mutation

```

001 package org.gxml.book.mutable;
002
003 import java.math.BigDecimal;
004
005 import javax.xml.XMLConstants;
006
007 import org.gxml.book.common.MutableApp;
008 import org.gxml.sa.GxAtomBridge;
009 import org.gxml.sa.GxNameBridge;
010 import org.gxml.sa.mutable.GxModelMutable;
011 import org.gxml.sa.mutable.GxProcessingContextMutable;
012 import org.gxml.xdm.NodeKind;
013
014 /**
015  * This sample illustrates the use of the optional mutability API.
016  *
017  * @author dholmes
018  *
019  * @param <I>
020  * @param <U>
021  * @param <N>
022  * @param <A>
023  * @param <S>
024  * @param <T>
025  * @param <X>
026  */
027 public abstract class MutableSample<I, U, N extends I, A extends I, S, T, X>
extends
MutableApp<I, U, N, A, S, T, X>
028 {
029     /**
030      * This is a test of basic mutability through the optional mutability API.
031      * Line 2
032      * Line 3
033      * Line 4 // OK
034      */
035     public void testIntroduction() throws Exception
036     {
037         final GxProcessingContextMutable<I, U, N, A, S, T, X> pcx =
newProcessingContext();

```

```

038     final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();
039     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
040
041     /* // Create a new document. */
042     final N documentNode = pcx.newDocument();
043
044     final GxModelMutable<N, A, S, T> model = pcx.getModel();
045
046     assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
047
048     // Every node in the tree has an owner which is a document node. /* OK
049     */
050     final N owner = model.getOwner(documentNode);
051
052     assertTrue(model.isSameNode(documentNode, owner));
053
054     final S namespaceURI = nameBridge.symbolize("http://www.example.com");
055     final S localName = nameBridge.symbolize("foo");
056     final String prefix = "x";
057     final N documentElement = model.createElement(owner, namespaceURI,
058     localName, prefix);
059
060     // Append the document element to the documentNode.
061     model.appendChild(documentNode, documentElement);
062
063     model.setNamespace(documentElement, prefix, namespaceURI);
064
065     model.setAttribute(documentElement, nameBridge.empty(),
066     nameBridge.symbolize("version"),
067     XMLConstants.DEFAULT_NS_PREFIX, atomBridge.wrapAtom(atomBridge.createDecimal
068     (BigDecimal.valueOf(2.7))));
069
070     // Append four text nodes to the document element.
071     model.appendChild(documentElement, model.createText(owner, "Hello"));
072     model.appendChild(documentElement, model.createText(owner, " "));
073     model.appendChild(documentElement, model.createText(owner, "World"));
074     model.appendChild(documentElement, model.createText(owner, "!"));
075
076     // Compress the four contiguous text nodes into a single text node.
077     model.normalize(documentNode);
078
079     @SuppressWarnings("unused")
080     final String strval = serialize(documentNode, pcx);
081     //System.out.println(strval);
082 }
083 }

```

Serialization

```

001 package org.gxml.book.mutable;
002
003 import java.math.BigDecimal;
004
005 import javax.xml.XMLConstants;
006
007 import org.gxml.book.common.MutableApp;
008 import org.gxml.sa.GxAtomBridge;
009 import org.gxml.sa.GxNameBridge;
010 import org.gxml.sa.mutable.GxModelMutable;
011 import org.gxml.sa.mutable.GxProcessingContextMutable;
012 import org.gxml.xdm.NodeKind;
013
014 /**
015  * This sample illustrates the use of the optional mutability API.
016  *
017  * @author dholmes
018  *
019  * @param <I>
020  * @param <U>
021  * @param <N>

```

```

022 * @param <A>
023 * @param <S>
024 * @param <T>
025 * @param <X>
026 */
027 public abstract class MutableSample<I, U, N extends I, A extends I, S, T, X>
extends
MutableApp<I, U, N, A, S, T, X>
028 {
029     /**
030      * This is a test of basic mutability through the optional mutability API.
031      * Line 2
032      * Line 3
033      * Line 4 // OK
034      */
035     public void testIntroduction() throws Exception
036     {
037         final GxProcessingContextMutable<I, U, N, A, S, T, X> pcx =
newProcessingContext();
038         final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();
039         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
040
041         /* // Create a new document. */
042         final N documentNode = pcx.newDocument();
043
044         final GxModelMutable<N, A, S, T> model = pcx.getModel();
045
046         assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
047
048         /* Every node in the tree has an owner which is a document node. */ OK
049         final N owner = model.getOwner(documentNode);
050
051         assertTrue(model.isSameNode(documentNode, owner));
052
053         final S namespaceURI = nameBridge.symbolize("http://www.example.com");
054         final S localName = nameBridge.symbolize("foo");
055         final String prefix = "x";
056         final N documentElement = model.createElement(owner, namespaceURI,
localName, prefix);
057
058         /* Append the document element to the documentNode.
059         model.appendChild(documentNode, documentElement);
060
061         model.setNamespace(documentElement, prefix, namespaceURI);
062
063         model.setAttribute(documentElement, nameBridge.empty(),
nameBridge.symbolize("version"),
XMLConstants.DEFAULT_NS_PREFIX, atomBridge.wrapAtom(atomBridge.createDecimal
(BigDecimal.valueOf(2.7))));
064
065         /* Append four text nodes to the document element.
066         model.appendChild(documentElement, model.createText(owner, "Hello"));
067         model.appendChild(documentElement, model.createText(owner, " "));
068         model.appendChild(documentElement, model.createText(owner, "World"));
069         model.appendChild(documentElement, model.createText(owner, "!"));
070
071         /* Compress the four contiguous text nodes into a single text node.
072         model.normalize(documentNode);
073
074         @SuppressWarnings("unused")
075         final String strval = serialize(documentNode, pcx);
076         //System.out.println(strval);
077     }
078 }

```

XPath

```

001 package org.gxml.book.xpath;
002

```

```

003 import org.gxml.book.common.SampleApp;
004 import org.gxml.sa.GxMetaBridge;
005 import org.gxml.sa.GxNameBridge;
006 import org.gxml.sa.GxProcessingContext;
007 import org.gxml.sa.GxVariantBridge;
008 import org.gxml.xdm.Emulation;
009 import org.gxml.xs.SmName;
010 import org.gxml.xs.SmNativeType;
011
012 import com.tibco.gxml.sa.api.common.lang.ExprResult;
013 import com.tibco.gxml.sa.api.common.lang.GxExpr;
014 import com.tibco.gxml.sa.api.common.lang.GxExprContextDynamicArgs;
015 import com.tibco.gxml.sa.api.common.lang.GxExprContextStaticArgs;
016 import com.tibco.gxml.sa.api.common.lang.GxFocus;
017 import com.tibco.gxml.sa.api.common.lang.GxLanguageToolKit;
018 import com.tibco.gxml.sa.processor.xquery.LanguageToolKit;
019 import com.tibco.gxmlsa.processor.org.exslt.math.ExsltMathFunctionGroup;
020
021 public abstract class XPathSample<I, U, N extends I, A extends I, S, T, X>
    extends
    SampleApp<I, U, N, A, S, T, X>
022 {
023     public void testGettingStarted() throws Exception
024     {
025         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
026
027         // For demonstration purposes, register the language toolkit with the
processing context.
028         pcx.register("xyz", new LanguageToolKit<I, U, N, A, S, T, X>(pcx));
029
030         @SuppressWarnings("unchecked")
031         // Immediately get back the registered processor.
032         GxLanguageToolKit<I, U, N, A, S, T, X> xtk = pcx.getProcessor("xyz",
GxLanguageToolKit.class);
033
034         final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg =
xtk.newStaticContextArgs();
035
036         final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
037
038         final ExprResult<I, U, N, A, S, T, X> prepared =
xtk.prepare("concat('Hello', ' ', ' ',
'World', '!')", metaBridge.emptyType(), sarg);
039         final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
040
041         final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg =
xtk.newDynamicContextArgs();
042
043         final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);
044
045         assertEquals("Hello, World!", strval);
046     }
047
048     public void testBindingVariables() throws Exception
049     {
050         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
051
052         final GxLanguageToolKit<I, U, N, A, S, T, X> xtk =
new LanguageToolKit<I, U, N, A, S, T, X>(pcx);
053
054         final GxExprContextStaticArgs<I, U, N, A, S, T, X> statArgs =
xtk.newStaticContextArgs();
055         statArgs.setEmulation(Emulation.MODERN);
056
057         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
058         final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"),
nameBridge);
059
060         final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
061         statArgs.bindVariableType(varName,

```

```

metaBridge.getType(SmNativeType.STRING));
062
063     final String es = "concat('Hello', ' ', ' ', $x, '!')";
064     final T sfocus = metaBridge.emptyType();
065
066     final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare(es,
sfocus, statArgs);
067
068     final GxExprContextDynamicArgs<I, U, N, A, S, T, X> dynArgs =
xtk.newDynamicContextArgs();
069     dynArgs.setEmulation(Emulation.MODERN);
070
071     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
072     final X value = valueBridge.stringValue("World");
073     dynArgs.bindVariableValue(varName, value);
074
075     final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
076     final GxFocus<I> dfocus = xtk.emptyFocus();
077     final String strval = expr.stringFunction(dfocus, dynArgs, pcx);
078
079     assertEquals("Hello, World!", strval);
080 }
081
082 public void testEXSLT() throws Exception
083 {
084     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
085     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
086
087     final GxLanguageToolKit<I, U, N, A, S, T, X> xtk =
new LanguageToolKit<I, U, N, A, S, T, X>(pcx);
088
089     final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg =
xtk.newStaticContextArgs();
090     sarg.getInScopeNamespaces().declarePrefix("math", nameBridge.symbolize
("http://exslt.org/math"));
091     final ExsltMathFunctionGroup<I, U, N, A, S, T, X>
exsltMathFunctionGroup = new
ExsltMathFunctionGroup<I, U, N, A, S, T, X>("http://exslt.org/math", pcx);
092     sarg.setFunctionSigns("http://exslt.org/math", exsltMathFunctionGroup);
093     // The function implementations can be provided now or just prior to
execution.
094     sarg.setFunctionImpls("http://exslt.org/math", exsltMathFunctionGroup);
095
096     final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
097
098     final ExprResult<I, U, N, A, S, T, X> prepared =
xtk.prepare("math:exp(1)",
metaBridge.emptyType(), sarg);
099
100     final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
101
102     final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg =
xtk.newDynamicContextArgs();
103     // Here we also (redundantly) provide the function implementations just
prior to execution.
104     darg.setFunctionImpls("http://exslt.org/math", exsltMathFunctionGroup);
105
106     final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);
107
108     assertEquals("2.7182818284590455", strval);
109 }
110
111 public void testExpressionType() throws Exception
112 {
113     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
114
115     final GxLanguageToolKit<I, U, N, A, S, T, X> xtk =
new LanguageToolKit<I, U, N, A, S, T, X>(pcx);
116
117     final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg =

```

```

xTk.newStaticContextArgs();
118
119         final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
120
121         final ExprResult<I, U, N, A, S, T, X> prepared = xTk.prepare("'Hello'",
metaBridge.emptyType(), sarg);
122         /* final GxExpr<I, U, N, A, S, T, X> expr = */prepared.getExpr();
123         /* final GxExprInfo<T> info = */prepared.getInfo();
124     }
125 }

```

XSLT

```

001 package org.gxml.book.xslt;
002
003 import java.io.IOException;
004 import java.io.InputStream;
005 import java.io.StringReader;
006 import java.io.StringWriter;
007 import java.net.URI;
008 import java.net.URISyntaxException;
009
010 import javax.xml.namespace.QName;
011 import javax.xml.parsers.ParserConfigurationException;
012
013 import org.gxml.book.common.SampleApp;
014 import org.gxml.sa.GxException;
015 import org.gxml.sa.GxMetaBridge;
016 import org.gxml.sa.GxModel;
017 import org.gxml.sa.GxNameBridge;
018 import org.gxml.sa.GxProcessingContext;
019 import org.gxml.sa.GxSequenceHandler;
020 import org.gxml.sa.GxVariantBridge;
021 import org.gxml.xdm.NodeKind;
022 import org.gxml.xdm.Resolved;
023 import org.gxml.xdm.Resolver;
024 import org.gxml.xs.SmName;
025 import org.gxml.xs.SmNativeType;
026
027 import com.tibco.gxml.sa.api.common.lang.ExprException;
028 import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
029 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
030 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
031 import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
032 import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
033 import com.tibco.gxml.sa.processor.xslt.GxTransform;
034 import com.tibco.gxml.sa.processor.xslt.GxTransformBuilder;
035 import com.tibco.gxml.sa.processor.xslt.GxTransformer;
036 import com.tibco.gxml.sa.processor.xslt.XSLTransformBuilder;
037 import com.tibco.gxmlsa.processor.org.exslt.strings.ExsltStringsFunctionGroup;
038
039 public abstract class XSLTSample<I, U, N extends I, A extends I, S, T, X>
extends SampleApp<I, U, N, A, S, T, X>
040 {
041     public void testExample() throws ParserConfigurationException, IOException,
GxException, ExprException, URISyntaxException
042     {
043         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
044         final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
045         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
046
047         final Resolver resolver = getResolver();
048
049         final URI xmlSystemId = new URI("hotel.xml");
050         final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);
051
052         final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I,
U, N, A, S, T, X>(pcx);

```

```

053         f.setIgnoreComments(false);
054
055         final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
056
057         final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
058
059         final URI xslSystemId = new URI("hotel.xsl");
060         final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);
061
062         final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
063
064         compiler.setCompatibleMode(true);
065         // compiler.setRestrictedMode(true); // XSLT 2.0 subset for mapper.
066
067         // Specify the static type for the context item:
068         // document-node(element(*,xs:untyped))
069         final T documentType =
metaBridge.documentType(metaBridge.elementType(new SmName<S>(null, null,
nameBridge), metaBridge.getType(SmNativeType.UNTYPED), false));
070         compiler.setFocus(documentType);
071
072         final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());
073
074         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
075
076         // TODO: Extract output configuration.
077         // compiled.configure(sf);
078
079         sf.setIndent(true);
080
081         final StringWriter w = new StringWriter();
082
083         final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
084
085         final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
086
087         transformer.transform(document, pcx, handler);
088
089         @SuppressWarnings("unused")
090         final String s = w.toString();
091         // System.out.println(s);
092     }
093
094     @SuppressWarnings("unused")
095     private void bar(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
096     {
097         try
098         {
099             final GxTransformBuilder<I, U, N, A, S, T, X> builder = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
100
101             final GxTransform<I, U, N, A, S, T, X> transform =
builder.prepareTransform(new StringReader("<x xsl:version='1.0' xmlns:xsl='http://
www.w3.org/1999/XSL/Transform'></x>"), new URI(""));
102
103             final GxTransformer<I, U, N, A, S, T, X> transformer =
transform.newTransformer();
104
105             final N document = transformer.transform(null, pcx);
106
107             final GxModel<N, A, S, T> model = pcx.getModel();
108
109             final N element = model.getFirstChild(document);
110
111             final String name = model.getLocalNameAsString(element);
112

```

```

113         // System.out.println("XSLT: " + name);
114     }
115     catch (final Throwable e)
116     {
117         e.printStackTrace();
118     }
119 }
120
121 public void skipVariableBinding() throws ParserConfigurationException,
IOException, GxException, ExprException, URISyntaxException
122 {
123     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
124
125     final Resolver resolver = getResolver();
126
127     final URI xslSystemId = new URI("email.xsl");
128     final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);
129
130     final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
131
132     final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());
133
134     final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
135
136     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
137     final SmName<S> varName = nameBridge.name(new QName("to"));
138     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
139     final X value = valueBridge.stringValue("David");
140
141     transformer.bindVariableValue(varName, value);
142     transformer.bindVariableValue(nameBridge.name(new QName("http://
www.example.com", "from")), valueBridge.stringValue("Julie"));
143
144     final N documentNode = transformer.transform(null, pcx);
145
146     final GxModel<N, A, S, T> model = pcx.getModel();
147
148     assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
149     final N email = model.getFirstChildElement(documentNode);
150     final N to = model.getFirstChildElementByName(email,
nameBridge.symbolize("http://www.example.com"), nameBridge.symbolize("to"));
151     assertEquals("David", model.getStringValue(to));
152     final N from = model.getFirstChildElementByName(email, null,
nameBridge.symbolize("from"));
153     assertEquals("Julie", model.getStringValue(from));
154     final N again = model.getFirstChildElementByName(email,
nameBridge.symbolize("http://www.example.com"), null);
155     assertEquals("David", model.getStringValue(again));
156 }
157
158 public void skipExternalFunctions() throws ParserConfigurationException,
IOException, GxException, ExprException, URISyntaxException
159 {
160     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
161
162     final Resolver resolver = getResolver();
163
164     final Resolved<InputStream> xmlInput = resolver.resolveInputStream(new
URI("exslt.xml"));
165
166     final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I,
U, N, A, S, T, X>(pcx);
167
168     final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
169
170     final N document = builder.parse(xmlInput.getResource(),

```

```

xmlInput.getSystemId());
171
172         final Resolved<InputStream> xslInput = resolver.resolveInputStream(new
URI("exslt.xsl"));
173
174         final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
175
176         final String namespaceURI = "http://exslt.org/strings";
177         final ExsltStringsFunctionGroup<I, U, N, A, S, T, X> functions = new
ExsltStringsFunctionGroup<I, U, N, A, S, T, X>(namespaceURI, pcx);
178         compiler.setFunctionSigns(namespaceURI, functions);
179         compiler.setFunctionImpls(namespaceURI, functions);
180
181         final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());
182
183         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
184
185         // TODO: Extract configuration.
186         // compiled.configure(sf);
187
188         sf.setIndent(true);
189
190         final StringWriter w = new StringWriter();
191
192         final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
193
194         final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
195
196         transformer.transform(document, pcx, handler);
197
198         // System.out.println(w.toString());
199     }
200
201     public void skipHotel() throws ParserConfigurationException, IOException,
GxException, ExprException, URISyntaxException
202     {
203         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
204
205         final Resolver resolver = getResolver();
206
207         final Resolved<InputStream> xmlInput = resolver.resolveInputStream(new
URI("hotel.xml"));
208
209         final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I,
U, N, A, S, T, X>(pcx);
210
211         final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
212
213         final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
214
215         final Resolved<InputStream> xslInput = resolver.resolveInputStream(new
URI("hotel.xsl"));
216
217         final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
218
219         final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());
220
221         final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
222         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
223         final SmName<S> varName = nameBridge.name(new QName("MessageData"));
224         final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
225         final X value = valueBridge.node(document);
226

```

```

227         transformer.bindVariableValue(varName, value);
228
229         final N documentNode = transformer.transform(null, pcx);
230
231         final GxModel<N, A, S, T> model = pcx.getModel();
232
233         assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
234         final N searchHotelRequest = model.getFirstChildElement(documentNode);
235         final N parameters =
model.getFirstChildElementByName(searchHotelRequest, nameBridge.symbolize("http://
xmlns.example.com/1189038295781"), nameBridge.symbolize("parameters"));
236         final N searchHotel = model.getFirstChildElementByName(parameters,
nameBridge.symbolize("http://www.xyzcorp/procureservice/QueryGDS_Europe/"),
nameBridge.symbolize("searchHotel"));
237         final N country = model.getFirstChildElementByName(searchHotel,
nameBridge.symbolize("http://www.xyzcorp/procureservice/QueryGDS_Europe/"),
nameBridge.symbolize("country"));
238         assertEquals("USA", model.getStringValue(country));
239     }
240 }

```

XQuery

```

001 package org.gxml.book.xquery;
002
003 import java.io.StringWriter;
004 import java.math.BigInteger;
005 import java.net.URI;
006
007 import javax.xml.namespace.QName;
008
009 import org.gxml.book.common.SampleApp;
010 import org.gxml.sa.GxAtomBridge;
011 import org.gxml.sa.GxNameBridge;
012 import org.gxml.sa.GxProcessingContext;
013 import org.gxml.sa.GxSequenceHandler;
014 import org.gxml.sa.GxVariantBridge;
015 import org.gxml.xs.SmName;
016
017 import com.tibco.gxml.sa.api.common.lang.GxXQConnection;
018 import com.tibco.gxml.sa.api.common.lang.GxXQDataSource;
019 import com.tibco.gxml.sa.api.common.lang.GxXQExpression;
020 import com.tibco.gxml.sa.api.common.lang.GxXQPreparedExpression;
021 import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
022 import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
023 import com.tibco.gxml.sa.processor.xquery.XQEngine;
024 import com.tibco.gxml.sa.processor.xquery.XQErrorCatcher;
025
026 /**
027  * Introduction to XQuery.
028  */
029 public abstract class XQuerySample<I, U, N extends I, A extends I, S, T, X>
extends SampleApp<I, U, N, A, S, T, X>
030 {
031     public void testExample() throws Exception
032     {
033         // Obtain a new processing context from the application.
034         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
035
036         final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A,
S, T, X>(pcx);
037
038         final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();
039
040         final String expression = "<x>{text{for $i in (1,2,3,4) return $i *
2}}</x>";
041
042         final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
conn.prepareExpression(expression);

```

```

043
044     final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
045     sf.setMethod(new QName("xml"));
046     sf.setOmitXmlDeclaration(true);
047     final StringWriter sw = new StringWriter();
048     final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);
049
050     expr.executeQuery(handler);
051
052     final String actual = sw.toString();
053     assertEquals(expression, "<x>2 4 6 8</x>", actual);
054 }
055
056 public void testGettingStarted() throws Exception
057 {
058     // Obtain a new processing context from the application.
059     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
060
061     final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A,
S, T, X>(pcx);
062
063     final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();
064
065     final GxXQExpression<I, U, N, A, S, T, X> expr =
conn.createExpression();
066
067     final String es = "for $n in fn:doc('catalog.xml')//item return
fn:data($n/name)";
068
069     final URI systemId = new URI("catalog.xml");
070
071     expr.setBaseURI(systemId);
072
073     @SuppressWarnings("unused")
074     final X value = expr.executeQuery(es);
075 }
076
077 public void testHelloWorld() throws Exception
078 {
079     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
this.newProcessingContext();
080
081     final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A,
S, T, X>(pcx);
082
083     final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();
084
085     conn.setScriptingMode(true);
086
087     final String expression = "declare variable $x external; concat('Hello,
', $x, '!!')";
088
089     final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
conn.prepareExpression(expression);
090
091     final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);
092     sf.setOmitXmlDeclaration(true);
093     sf.setIndent(false);
094     sf.setMethod(new QName("xml"));
095     final StringWriter sw = new StringWriter();
096     final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);
097
098     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
099     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
100
101     final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"),
nameBridge);
102     final X value = valueBridge.stringValue("World");
103

```

```

104         expr.bindVariableValue(varName, value);
105
106         expr.executeQuery(handler);
107
108         String actual = sw.toString();
109         assertEquals(expression, "Hello, World!", actual);
110     }
111
112     public void testMergeTextNodes() throws Exception
113     {
114         // Obtain a new processing context from the application.
115         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
116 newProcessingContext();
117         final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A,
118 S, T, X>(pcx);
119         final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();
120
121         // final String expression = "";
122         final String expression = "count((element elem {1, 'string', 1,2e3})/
123 text())";
124         final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
125 conn.prepareExpression(expression);
126         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
127 SerializerFactory<I, U, N, A, S, T, X>(pcx);
128         sf.setMethod(new QName("xml"));
129         sf.setOmitXmlDeclaration(true);
130         final StringWriter sw = new StringWriter();
131         final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);
132
133         expr.executeQuery(handler);
134
135         final String actual = sw.toString();
136         assertEquals(expression, "1", actual);
137     }
138
139     public void testProblem() throws Exception
140     {
141         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
142 this.newProcessingContext();
143         final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A,
144 S, T, X>(pcx);
145         final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();
146
147         final XQErrorCatcher messages = new XQErrorCatcher();
148
149         conn.setErrorHandler(messages);
150         conn.setCompatibleMode(false);
151         conn.setScriptingMode(true);
152
153         final String expression =
154 "(xs:untypedAtomic('1'),xs:untypedAtomic('2')) = (xs:untypedAtomic('2.0'),2.0)";
155         final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
156 conn.prepareExpression(expression);
157
158         final X value = expr.executeQuery();
159
160         final GxVariantBridge<I, N, A, X> variantBridge =
161 pcx.getVariantBridge();
162         switch (variantBridge.getNature(value))
163         {
164             case ITEMS:
165             {
166                 @SuppressWarnings("unused")
167                 final Iterable<I> items = variantBridge.getItemSet(value);
168                 // System.out.println(items);
169             }
170         }
171     }

```

```

166         }
167         break;
168         case ATOM:
169         {
170             @SuppressWarnings("unused")
171             final A atom = variantBridge.getAtom(value);
172             @SuppressWarnings("unused")
173             final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();
174             // System.out.println(atomBridge.getC14NForm(atom));
175         }
176         break;
177         case STRING:
178         {
179             @SuppressWarnings("unused")
180             final String strval = variantBridge.getString(value);
181             // System.out.println(strval);
182         }
183         break;
184         case INTEGER:
185         {
186             @SuppressWarnings("unused")
187             final BigInteger integer = variantBridge.getInteger(value);
188             // System.out.println(integer);
189         }
190         break;
191         default:
192         {
193             throw new AssertionError(variantBridge.getNature(value));
194         }
195     }
196 }
197
198 public void testTyping() throws Exception
199 {
200     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
201     this.newProcessingContext();
202     final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A,
203     S, T, X>(pcx);
204     final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();
205
206     conn.setScriptingMode(true);
207
208     final XQErrorCatcher messages = new XQErrorCatcher();
209
210     conn.setErrorHandler(messages);
211
212     final String expression = "declare variable $x external;
213     contains(string(number($x)), 'NaN')";
214     final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
215     conn.prepareExpression(expression);
216     final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
217     SerializerFactory<I, U, N, A, S, T, X>(pcx);
218     sf.setOmitXmlDeclaration(true);
219     sf.setIndent(false);
220     sf.setMethod(new QName("xml"));
221     final StringWriter sw = new StringWriter();
222     final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);
223
224     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
225     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
226     final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"),
227     nameBridge);
228     final X value = valueBridge.doubleValue(5.0);
229
230     expr.bindVariableValue(varName, value);
231
232     expr.executeQuery(handler);

```

```

232
233     String actual = sw.toString();
234     assertEquals(expression, "false", actual);
235 }
236 }

```

Validation

```

001 package org.gxml.book.validation;
002
003 import java.io.InputStream;
004 import java.net.URI;
005 import java.util.LinkedList;
006 import java.util.List;
007
008 import org.gxml.book.common.SampleApp;
009 import org.gxml.sa.GxFragmentBuilder;
010 import org.gxml.sa.GxModel;
011 import org.gxml.sa.GxProcessingContext;
012 import org.gxml.xdm.Resolved;
013 import org.gxml.xdm.Resolver;
014 import org.gxml.xs.SmException;
015 import org.gxml.xs.SmExceptionCatcher;
016 import org.gxml.xs.SmExceptionHandler;
017 import org.gxml.xs.SmMetaLoadArgs;
018
019 import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
020 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
021 import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
022 import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
023 import com.tibco.gxml.sa.processor.validation.GxContentValidator;
024 import com.tibco.gxml.sa.processor.validation.GxValidatorCache;
025 import com.tibco.gxml.sa.processor.validation.GxValidatorCacheFactory;
026 import com.tibco.gxml.sa.processor.validation.ValidatorCacheFactory;
027 import com.tibco.gxml.xs.W3cXmlSchemaParser;
028
029 public abstract class ValidationSample<I, U, N extends I, A extends I, S, T, X>
    extends SampleApp<I, U, N, A, S, T, X>
030 {
031     public void testByteStreamValidation() throws Exception
032     {
033         // Load a top-level schema into the processing context.
034         final List<Resolved<InputStream>> resources = new
    LinkedList<Resolved<InputStream>>();
035         resources.add(getResolver().resolveInputStream(new
    URI("PurchaseOrder.xsd")));
036
037         final SmExceptionCatcher errors = new SmExceptionCatcher();
038         final SmMetaLoadArgs args = new SmMetaLoadArgs();
039
040         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
    newProcessingContext();
041
042         final W3cXmlSchemaParser<A, S> parser = new W3cXmlSchemaParser<A,
    S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A, S>(pcx.getAtomBridge()));
043
044         for (final Resolved<InputStream> resource : resources)
045         {
046             pcx.register(parser.parse(resource.getLocation(),
    resource.getResource(), resource.getSystemId(), errors, args, pcx));
047         }
048
049         pcx.lock();
050
051         // Create a validator...
052         final GxValidatorCacheFactory<A, S, T> vcf = new
    ValidatorCacheFactory<I, U, N, A, S, T, X>(pcx);
053         final GxValidatorCache<A, S, T> vc = vcf.newValidatorCache();
054         final GxContentValidator<A, S, T> validator = vc.newContentValidator();
055

```

```

056         // Set the downstream event handler which contains annotations and
typed content.
057         // validator.setGxContentHandler(/* ...*/null);
058         validator.setExceptionHandler(errors);
059
060         // The document node that we wish to validate.
061         final Resolved<InputStream> xmlInput =
getResolver().resolveInputStream(new URI("PurchaseOrder.xml"));
062
063         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
064
065         final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
066
067         final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
068
069         // Stream the document into the validator.
070         final GxModel<N, A, S, T> model = pcx.getModel();
071
072         model.stream(document, true, true, validator);
073
074         if (errors.size() > 0)
075         {
076             // You've got errors.'
077         }
078     }
079
080     public void testTreeValidation() throws Exception
081     {
082         final Resolver resolver = getResolver();
083
084         // Load a top-level schema into the processing context.
085         final List<Resolved<InputStream>> resources = new
LinkedList<Resolved<InputStream>>();
086         resources.add(getResolver().resolveInputStream(new
URI("PurchaseOrder.xsd")));
087
088         final SmExceptionCatcher errors = new SmExceptionCatcher();
089         final SmMetaLoadArgs args = new SmMetaLoadArgs();
090
091         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
092         final W3cXmlSchemaParser<A, S> parser = new W3cXmlSchemaParser<A,
S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A, S>(pcx.getAtomBridge()));
093         for (final Resolved<InputStream> resource : resources)
094         {
095             pcx.register(parser.parse(resource.getLocation(),
resource.getResource(), resource.getSystemId(), errors, args, pcx));
096         }
097         pcx.lock();
098         // The document node that we wish to validate.
099         @SuppressWarnings("unused")
100         final URI xmlLocation = new URI("PurchaseOrder.xml");
101         final URI xmlSystemId = new URI("PurchaseOrder.xml");
102         final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);
103
104         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
105
106         final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
107
108         final N documentIn = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());
109
110         @SuppressWarnings("unused")
111         final N documentOut = validate(documentIn, errors, pcx);
112
113         if (errors.size() > 0)
114         {
115             // You've got errors.'

```

```

116         for (@SuppressWarnings("unused")
117             final SmException error : errors)
118         {
119             // System.out.println(error.getLocalizedMessage());
120         }
121     }
122 }
123
124 /**
125  * This static function illustrates a helper function for validating a
126  * document tree. <br/>
127  * Note that we assume that the processing context is already loaded with
128  * meta-data.
129  *
130  * @param node
131  *         The input document.
132  * @param errors
133  *         The error handler.
134  * @param pcx
135  *         The processing context.
136  */
137 public static <I, U, N extends I, A extends I, S, T, X> N validate(final N
138 node, final SmExceptionHandler errors, final GxProcessingContext<I, U, N, A, S, T,
139 X> pcx)
140 {
141     final GxValidatorCacheFactory<A, S, T> vcf = new
142     ValidatorCacheFactory<I, U, N, A, S, T, X>(pcx);
143
144     // We already have a tree as input so we'll use the content validator'
145     // and stream the document in as a bunch of events (a bit like SAX, but
146     not lexical).
147     final GxValidatorCache<A, S, T> vc = vcf.newValidatorCache();
148
149     final GxContentValidator<A, S, T> validator = vc.newContentValidator();
150
151     validator.setExceptionHandler(errors);
152
153     final GxModel<N, A, S, T> model = pcx.getModel();
154
155     // We want to produce a node so we'll need a fragment builder at the
156     output.'
157     final GxFragmentBuilder<N, A, S, T> builder = pcx.newFragmentBuilder();
158
159     // Connect the pieces together so that the validation output builds a
160     tree.
161     validator.setGxContentHandler(builder);
162
163     // Make it so!
164     model.stream(node, true, true, validator);
165
166     // Practice safe coding: We don't know what might happen if there are
167     errors.'
168     final List<? extends N> nodes = builder.getNodes();
169     if (nodes.size() > 0)
170     {
171         return nodes.get(0);
172     }
173     else
174     {
175         return null;
176     }
177 }

```