

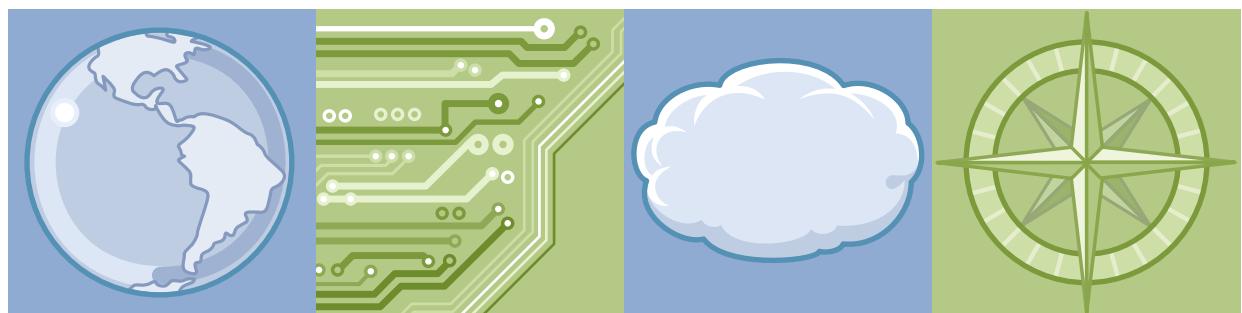


# IBM Training

Student Notebook

## IBM Integration Bus V10 Application Development II

Course code WM676/ZM676 ERC 1.0



## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

CICS®	DataPower®	DB™
DB2®	developerWorks®	Express®
IMS™	PartnerWorld®	SurePOS™
Tivoli®	WebSphere®	Worklight®
z/OS®		

Intel and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the “Marks”) of VMware, Inc. in the United States and/or other jurisdictions.

Sterling Commerce® is a trademark or registered trademark of IBM International Group B.V., an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

### March 2016 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

# Contents

<b>Trademarks .....</b>	<b>xv</b>
<b>Course description .....</b>	<b>xvii</b>
<b>Agenda .....</b>	<b>xix</b>
<b>Unit 1. Using event-driven processing nodes .....</b>	<b>1-1</b>
Unit objectives .....	1-2
How to check online for course material updates .....	1-3
Event-driven processing nodes .....	1-4
Topics .....	1-5
1.1. Message aggregation .....	1-7
Message aggregation .....	1-8
Message aggregation details .....	1-9
Aggregate Control node .....	1-10
Aggregate Control node configuration .....	1-11
Aggregate Request node .....	1-12
Aggregate Reply node .....	1-13
Implementing message aggregation .....	1-14
Aggregation message tree .....	1-15
Message aggregation example: Fan-out flow .....	1-16
Message aggregation example: Fan-in flow .....	1-17
Multiple units of work during aggregation .....	1-18
Aggregation implementation tips (1 of 2) .....	1-19
Aggregation implementation tips (2 of 2) .....	1-21
1.2. Message collection .....	1-23
Message collection .....	1-24
Message collection message tree .....	1-25
Collector node .....	1-27
1.3. Controlling message sequence .....	1-29
The message sequencing problem .....	1-30
Sequence node .....	1-31
Sequence node properties .....	1-32
Resequence node .....	1-33
Resequence node storage queues .....	1-35
Resequence node properties .....	1-36
Resequence node use cases .....	1-38
Using the nodes as a pair .....	1-39
1.4. Timeout nodes .....	1-41
Timeout nodes .....	1-42
Storage queues for Timeout nodes .....	1-43
Stand-alone (automatic) Timeout Notification .....	1-44
TimeoutRequest message .....	1-46
Timeout nodes example .....	1-48
Unit summary .....	1-50
Checkpoint questions .....	1-51
Checkpoint answers .....	1-52
Exercise 1 .....	1-53

Exercise objectives .....	1-54
<b>Unit 2. Transforming data with Microsoft .NET .....</b>	<b>2-1</b>
Unit objectives .....	2-2
Microsoft .NET framework overview .....	2-3
.NET common language runtime (CLR) .....	2-4
From source code to byte code .....	2-5
Running .NET in your enterprise .....	2-6
Integrating .NET with IBM Integration Bus .....	2-7
Scope of .NET code in IBM Integration Bus .....	2-8
.NET application domains in IBM Integration Bus .....	2-9
.NETCompute node .....	2-10
.NETCompute node configuration .....	2-11
Node templates for Visual Studio .....	2-12
Debug .NET code with the Visual Studio Debugger .....	2-13
Handling exceptions .....	2-14
Configurable services for .NET (1 of 2) .....	2-15
Configurable services for .NET (2 of 2) .....	2-16
IBM Integration Bus .NET API and documentation .....	2-18
Deploying a .NET assembly .....	2-19
Unit summary .....	2-20
Checkpoint questions .....	2-21
Checkpoint answers .....	2-22
<b>Unit 3. Transforming data with XSL stylesheets .....</b>	<b>3-1</b>
Unit objectives .....	3-2
XSL Transform node .....	3-3
XSL Transform node properties .....	3-4
Deployed and non-deployed stylesheets .....	3-6
Before you deploy stylesheets .....	3-7
Stylesheet search order .....	3-8
Working with XML messages and the logical message tree .....	3-9
Determining the output message format .....	3-10
Unit summary .....	3-11
Checkpoint questions .....	3-12
Checkpoint answers .....	3-13
<b>Unit 4. Analyzing XML documents .....</b>	<b>4-1</b>
Unit objectives .....	4-2
Using XML data .....	4-3
Data analysis overview .....	4-4
Data analysis process .....	4-5
Creating a Data Analysis project .....	4-6
IBM Integration Bus Healthcare Pack Data Analysis profiles .....	4-7
Data Analysis Project Overview editor .....	4-8
Analyzing documents .....	4-9
Data Analysis model .....	4-10
Data Filter view .....	4-11
Depth of descendants .....	4-12
Highlight all coexisting elements .....	4-13
XML Glossary File Lookup Service .....	4-14

Refining a target model .....	4-15
Creating Data Analysis tools .....	4-16
Generated subflow .....	4-17
Target model: Relational database map .....	4-18
Generated subflow that includes Database nodes .....	4-19
Unit summary .....	4-20
Checkpoint questions .....	4-21
Checkpoint answers .....	4-22
<b>Unit 5. Modeling complex data with DFDL.....</b>	<b>5-1</b>
Unit objectives .....	5-2
Topics .....	5-3
5.1. DFDL model review .....	5-5
DFDL data support (1 of 2) .....	5-6
DFDL data support (2 of 2) .....	5-7
DFDL object model .....	5-8
DFDL properties .....	5-9
DFDL expressions .....	5-10
5.2. Modeling complex data .....	5-11
Understanding the logical structure of the data .....	5-12
Configuring the DFDL annotations (1 of 2) .....	5-14
Configuring the DFDL annotations (2 of 2) .....	5-16
Configuring the DFDL annotations example .....	5-17
DFDL points of uncertainty .....	5-19
DFDL points of uncertainty example .....	5-20
Adding a discriminator to a DFDL element .....	5-21
Adding asserts .....	5-22
Adding an assert to DFDL element .....	5-23
Asserts with recoverable errors .....	5-24
Length prefixes .....	5-25
Length prefixes example .....	5-26
Parsed arrays .....	5-27
Direct dispatch choice .....	5-28
Unordered sequences .....	5-29
Adding a reference to another schema (1 of 3) .....	5-30
Adding a reference to another schema (2 of 3) .....	5-31
Adding a reference to another schema (3 of 3) .....	5-32
5.3. Industry-standard models .....	5-33
DFDL schemas for industry formats .....	5-34
DFDL for ISO 8583 .....	5-35
DFDL for NACHA .....	5-36
DFDL for EDIFACT .....	5-37
DFDL for HL7 v2 .....	5-38
DFDL for TLOG .....	5-39
Useful DFDL links .....	5-40
Unit summary .....	5-41
Checkpoint questions .....	5-42
Checkpoint answers .....	5-43
Exercise 2 .....	5-44
Exercise objectives .....	5-45

<b>Unit 6. Working with message sets and the MRM domain .....</b>	<b>6-1</b>
Unit objectives .....	6-2
Topics .....	6-3
6.1. The MRM message set .....	6-5
MRM and message set development .....	6-6
Enabling message set development .....	6-7
Specification, model, logical message tree .....	6-8
Message set project .....	6-9
Creating a message definition file .....	6-10
Message definition content .....	6-11
Message model objects .....	6-13
Message model objects properties .....	6-14
Message set: Container and unit of administration .....	6-15
Physical representation .....	6-17
Custom wire format (binary or CWF) .....	6-18
Logical message and CWF format .....	6-19
Tagged/Delimited String Format (text or TDS) .....	6-20
Logical message and TDS format .....	6-22
Logical message and XML format .....	6-23
Default and fixed values for elements .....	6-25
MRM message model versus XML schema .....	6-26
Create and populate message definition .....	6-27
Bottom-up approach for message definition .....	6-28
Setting physical properties .....	6-29
Value constraints .....	6-30
6.2. Referencing the MRM message in a message flow .....	6-31
Assigning the parser to a message (1 of 2) .....	6-32
Assigning a parser to a message (2 of 2) .....	6-33
MRM parser operation .....	6-34
Logical message tree: Root .....	6-35
Sample Compute node ESQL for MRM .....	6-36
Unit summary .....	6-37
Checkpoint questions .....	6-38
Checkpoint answers .....	6-39
<b>Unit 7. Supporting web services .....</b>	<b>7-1</b>
Unit objectives .....	7-2
Topics .....	7-3
7.1. Developing web service and web service clients in Integration Bus .....	7-5
IBM Integration Bus and web services .....	7-6
Approaches to developing web services .....	7-7
Types of web services .....	7-8
When to use SOAP nodes? When to use HTTP nodes? .....	7-9
HTTP listeners .....	7-10
7.2. HTTP nodes .....	7-11
HTTP message processing nodes .....	7-12
Message flow as HTTP/HTTPS client .....	7-13
Message flow as service provider .....	7-15
HTTP nodes main properties .....	7-16
7.3. SOAP nodes .....	7-17
SOAP message processing nodes .....	7-18

SOAP Input and SOAP Reply nodes .....	7-19
SOAP Input and SOAP Reply nodes example .....	7-20
SOAP Request node .....	7-21
SOAP AsyncRequest and SOAP AsyncResponse .....	7-22
Configuring SOAP AsyncRequest nodes .....	7-23
WSDL importer .....	7-24
Import WSDL into workspace .....	7-25
Expose a web service: Generated message flow .....	7-26
Expose a web service: Generated subflow .....	7-28
Invoke a web service: Generated message flow .....	7-29
Invoke a web service: Generated subflow .....	7-30
Configuring SOAP nodes .....	7-31
Generating WSDL from message set .....	7-33
Common SOAP message tree .....	7-34
SOAP message tree .....	7-36
SOAP Extract node .....	7-37
SOAP Envelope node .....	7-38
<b>7.4. WS-Addressing and WS-Security .....</b>	<b>7-39</b>
WS-Addressing .....	7-40
WS-Addressing asynchronous consumer scenario .....	7-42
What is WS-Security? .....	7-43
Web services and security .....	7-44
Transport level security and message security .....	7-45
WS-Security <Security> element .....	7-47
WS-Security authentication in IBM Integration Bus .....	7-48
WS-Security message level protection .....	7-49
WS-Security message part protection .....	7-50
WS-Security: Provider scenario .....	7-51
WS-Security: Consumer scenario .....	7-52
Policy sets .....	7-53
Administrative versus development responsibilities .....	7-54
Defining message part protection on the node .....	7-55
Assigning policy sets with flows and nodes .....	7-57
Assigning policy sets and bindings in the BAR file editor .....	7-58
Unit summary .....	7-59
Checkpoint questions (1 of 2) .....	7-60
Checkpoint questions (2 of 2) .....	7-61
Checkpoint answers (1 of 2) .....	7-62
Checkpoint answers (2 of 2) .....	7-63
Exercise 3 .....	7-64
Exercise objectives .....	7-65
<b>Unit 8. Developing integration solutions by using integration services.....</b>	<b>8-1</b>
Unit objectives .....	8-2
Topics .....	8-3
<b>8.1. Services on IBM Integration Bus .....</b>	<b>8-5</b>
What is a service? .....	8-6
Integration challenges in a service-oriented world .....	8-7
Understanding services through service discovery .....	8-8
Exchanging data through connectors .....	8-9
Control operational behavior through policies .....	8-10

Integration Registry .....	8-11
Integrating services with Integration Bus .....	8-12
<b>8.2. Developing integration services.....</b>	<b>8-13</b>
What is an integration service? .....	8-14
Approaches to building web services .....	8-15
Integration service project structure .....	8-16
Integration Service editor .....	8-17
Service Interface editor .....	8-18
Implementing service operation with flows .....	8-19
Handling errors with flows .....	8-20
<b>8.3. Deploying and testing integration services .....</b>	<b>8-21</b>
Deploy the integration service .....	8-22
Testing the integration service (1 of 2) .....	8-23
Testing the integration service (2 of 2) .....	8-24
Viewing message flow test results .....	8-25
<b>8.4. Generating a JavaScript client API .....</b>	<b>8-27</b>
JavaScript client API .....	8-28
JavaScript API client restrictions .....	8-29
Work flow for Browser client .....	8-30
Work flow for Node.js client .....	8-31
Generate JavaScript client API for an integration service .....	8-32
Deploy the integration service .....	8-33
View the generated resources .....	8-34
Generated resources .....	8-35
Unit summary .....	8-36
Checkpoint questions .....	8-37
Checkpoint answers .....	8-38
Exercise 4 .....	8-39
Exercise objectives .....	8-40

<b>Unit 9. Connecting a database by using a discovered service .....</b>	<b>9-1</b>
Unit objectives .....	9-2
Service discovery .....	9-3
Database services .....	9-4
Starting the Database Service editor .....	9-5
Database Service editor: Description .....	9-6
Database Service editor: Database Definition .....	9-7
Database Service editor: Select Resources .....	9-8
Database Service editor: Service Operations (1 of 3) .....	9-9
Database Service editor: Service Operations (2 of 3) .....	9-10
Database Service editor: Service Operations (3 of 3) .....	9-11
Database query details .....	9-12
Database Service editor: Service Interface .....	9-13
Database Service editor: Summary .....	9-14
Generated WSDL .....	9-15
Generated XSD .....	9-16
Database service in the Application Development view .....	9-17
Calling a database service from a Compute node (1 of 2) .....	9-18
Calling a database service from a Compute node (2 of 2) .....	9-19
Generated ESQL code for service operation .....	9-20
Database service operation error handling .....	9-22

Unit summary .....	9-23
Checkpoint questions .....	9-24
Checkpoint answers .....	9-25
<b>Unit 10. Connecting IBM MQ by using a discovered service.....</b>	<b>10-1</b>
Unit objectives .....	10-2
IBM MQ service .....	10-3
Starting the IBM MQ Service editor .....	10-4
IBM MQ Service editor: Description .....	10-5
IBM MQ Service editor: Connect .....	10-6
IBM MQ Service editor: Select Resources .....	10-7
IBM MQ Service editor: MQ Configuration .....	10-8
IBM MQ Service editor: Service Interface .....	10-9
IBM MQ Service editor: Summary .....	10-10
IBM MQ Service WSDL and XSD file .....	10-11
Implementing an IBM MQ service .....	10-12
Calling an IBM MQ service .....	10-13
Integration Registry .....	10-14
Discover and catalog services .....	10-15
Publishing a service to the Integration Registry .....	10-16
Importing a service from the Integration Registry .....	10-17
Unit summary .....	10-18
Checkpoint questions .....	10-19
Checkpoint answers .....	10-20
Exercise 5 .....	10-21
Exercise objectives .....	10-22
<b>Unit 11. Creating a decision service.....</b>	<b>11-1</b>
Unit objectives .....	11-2
IBM Operational Decision Manager .....	11-3
Decision services in Integration Bus .....	11-4
Integration with Operation Decision Manager .....	11-5
Structure of a business rule .....	11-6
Steps for implementing a decision service .....	11-8
Integration Toolkit Decision Service editor .....	11-9
Decision Service node .....	11-10
Stock Trade example: Message flow .....	11-12
Stock Trade example: Message model .....	11-13
Stock Trade example: Define the decision service (1 of 2) .....	11-14
Stock Trade example: Define the decision service (2 of 2) .....	11-15
Stock Trade example: Add decision service rules (1 of 2) .....	11-16
Stock Trade example: Add decision service rules (2 of 2) .....	11-17
Stock Trade example: Embed the decision service .....	11-18
Stock Trade example: Configure the Decision Service node .....	11-19
Stock Trade example: Complete the message flow .....	11-20
Using a rule application archive .....	11-21
Importing a rule application archive .....	11-23
DecisionServiceRepository configurable service .....	11-24
Monitoring a decision service .....	11-25
Unit summary .....	11-26
Checkpoint questions .....	11-27

Checkpoint answers .....	11-28
Exercise 6 .....	11-29
Exercise objectives .....	11-30
<b>Unit 12. Developing integration solutions by using a REST API.....</b>	<b>12-1</b>
Unit objectives .....	12-2
REST API resources .....	12-3
REST API operations .....	12-4
REST API parameters .....	12-5
Swagger .....	12-6
Integration Bus support for Swagger documents .....	12-7
Creating a REST API in Integration Bus .....	12-8
Creating a REST API .....	12-9
REST API Description view .....	12-10
Implementing an operation .....	12-11
Accessing REST parameters: Programming examples .....	12-12
Accessing REST parameters in a graphical data map .....	12-13
Mapping JSON request bodies .....	12-14
Packaging and deploying REST APIs .....	12-15
Finding the base URL of the REST API .....	12-16
Deployed Swagger documents .....	12-17
Cross-Origin Resource Sharing (CORS) .....	12-18
REST API administration .....	12-19
Unit summary .....	12-21
Checkpoint questions .....	12-22
Checkpoint answers .....	12-23
<b>Unit 13. Using the global cache.....</b>	<b>13-1</b>
Unit objectives .....	13-2
IBM Integration Bus global cache .....	13-3
Scenario 1: Storing state for integrations .....	13-5
Scenario 2: Caching infrequently changing data .....	13-6
Global cache components .....	13-7
Integration Bus default cache topology .....	13-8
Global cache policy files .....	13-9
Integration node global cache properties .....	13-10
Connectivity to WebSphere eXtreme Scale grids .....	13-12
Access the global cache by using a JavaCompute node .....	13-13
Access data in the cache by using a JavaCompute node .....	13-14
Access the global cache by using a Mapping node .....	13-15
Providing values for key-value pairs in Cache Put .....	13-16
Provide a value for the Cache Get key .....	13-17
Global cache administrative tools .....	13-18
Unit summary .....	13-19
Checkpoint questions .....	13-20
Checkpoint answers .....	13-21
<b>Unit 14. Implementing message flow security.....</b>	<b>14-1</b>
Unit objectives .....	14-2
Topics .....	14-3
14.1. Security overview.....	14-5

IBM Integration Bus security overview . . . . .	14-6
IBM Integration Bus administration security . . . . .	14-8
Message transport security . . . . .	14-10
Message flow security overview . . . . .	14-11
<b>14.2. Message security. . . . .</b>	<b>14-13</b>
Security-enabled message processing nodes . . . . .	14-14
Message flow security manager . . . . .	14-16
Security profiles . . . . .	14-18
SecurityProfiles configurable service . . . . .	14-20
Display security profile properties . . . . .	14-22
Setting security profiles in the BAR file . . . . .	14-23
External security providers . . . . .	14-24
Security processing details: Flow diagram . . . . .	14-26
Security processing details (1 of 4) . . . . .	14-27
Security processing details (2 of 4) . . . . .	14-28
Security processing details (3 of 4) . . . . .	14-29
Security processing details (4 of 4) . . . . .	14-30
More about security processing . . . . .	14-31
Using the SecurityPEP node . . . . .	14-32
Security exception handling . . . . .	14-34
Diagnosing security problems . . . . .	14-35
Unit summary . . . . .	14-36
Checkpoint questions (1 of 2) . . . . .	14-37
Checkpoint questions (2 of 2) . . . . .	14-38
Checkpoint answers (1 of 2) . . . . .	14-39
Checkpoint answers (2 of 2) . . . . .	14-40
Exercise 7 . . . . .	14-41
Exercise objectives . . . . .	14-42
<b>Unit 15. Implementing publish/subscribe . . . . .</b>	<b>15-1</b>
Unit objectives . . . . .	15-2
Topics . . . . .	15-3
<b>15.1. Introduction to publish/subscribe . . . . .</b>	<b>15-5</b>
What is publish/subscribe? . . . . .	15-6
Topics and filters . . . . .	15-7
Topics and filters: Example . . . . .	15-8
Topic specifications . . . . .	15-9
Integration Bus and publish/subscribe support (1 of 2) . . . . .	15-11
Integration Bus and publish/subscribe support (2 of 2) . . . . .	15-12
Viewing publish/subscribe properties . . . . .	15-13
<b>15.2. Publish/subscribe with IBM MQ . . . . .</b>	<b>15-15</b>
IBM MQ publish/subscribe . . . . .	15-16
Managing publish/subscribe with IBM MQ Explorer . . . . .	15-17
IBM Integration Bus and IBM MQ . . . . .	15-18
Publish/subscribe message flows and applications . . . . .	15-19
Registering subscriptions in IBM Integration Bus (1 of 2) . . . . .	15-20
Registering subscription in IBM Integration Bus (2 of 2) . . . . .	15-21
Publication node . . . . .	15-23
Publication flows . . . . .	15-24
Content-based filtering . . . . .	15-26
Content-based filtering configuration . . . . .	15-27

Configure content-based filtering in Integration web interface .....	15-28
<b>15.3. Publish/subscribe with MQTT .....</b>	<b>15-29</b>
MQTT overview .....	15-30
MQTT administration with IBM MQ Explorer .....	15-31
MQTT support in IBM Integration Bus .....	15-32
Configuring the Integration Bus MQTT broker .....	15-33
Support for MQTT in a message flow .....	15-34
MQTT operational policies .....	15-35
Creating an MQTT policy in the Integration Toolkit .....	15-36
Updating an attached MQTT policy .....	15-37
Unit summary .....	15-38
Checkpoint questions .....	15-39
Checkpoint answers .....	15-40
<b>Unit 16. Monitoring message flow events .....</b>	<b>16-1</b>
Unit objectives .....	16-2
Topics .....	16-3
<b>16.1. Monitoring events .....</b>	<b>16-5</b>
Event monitoring and auditing .....	16-6
Event message categories .....	16-7
Configuring the publication of event messages .....	16-8
Subscribing to business event messages .....	16-10
Message flow event monitoring configuration .....	16-11
Event monitoring support: Input node .....	16-12
Event monitoring support: Output node .....	16-13
Adding a monitoring event to a node (1 of 3) .....	16-14
Adding a monitoring event to a node (2 of 3) .....	16-15
Adding a monitoring event to a node (3 of 3) .....	16-16
Customizing a monitoring event: Bitstream data .....	16-17
Event filter .....	16-18
Use an XPath expression to filter event emission .....	16-19
Customizing a message flow event: Correlation .....	16-20
Transaction correlators .....	16-21
Unit of work .....	16-22
Message flow event monitoring profiles .....	16-24
Example: Message flow event monitoring profile .....	16-25
Apply a message flow monitoring profile to a BAR file .....	16-26
<b>16.2. Record and replay .....</b>	<b>16-27</b>
Record and replay .....	16-28
Record and replay configuration .....	16-29
Preparing to record messages .....	16-30
Creating the database for recorded messages .....	16-31
Viewing the recorded data .....	16-32
IBM Integration web interface: Data viewer tab .....	16-33
Replaying messages .....	16-34
Unit summary .....	16-35
Checkpoint questions .....	16-36
Checkpoint answers .....	16-37
Exercise 8 .....	16-38
Exercise objectives .....	16-39

<b>Unit 17. Managing the workload .....</b>	<b>17-1</b>
Unit objectives .....	17-2
Control the processing speed in Integration Bus .....	17-3
Message flow processing rate control terms .....	17-4
Message flow processing rate control: Notification .....	17-5
Message flow processing rate control: Delay .....	17-7
Unresponsive flows .....	17-9
Handling unresponsive flows .....	17-10
Workload Management properties .....	17-11
Processing Timeout notification topics .....	17-12
Processing Timeout alert example .....	17-14
Workload Management policies in the Integration Registry .....	17-15
Using the Integration web interface to manage policies .....	17-16
Using the Integration web interface to attach a policy .....	17-17
Unit summary .....	17-18
Checkpoint questions .....	17-19
Checkpoint answers .....	17-20
<b>Unit 18. Creating patterns for reusability .....</b>	<b>18-1</b>
Unit objectives .....	18-2
Patterns overview .....	18-3
Using patterns from the GitHub repository .....	18-4
User-defined patterns overview .....	18-5
Process for creating a user-defined pattern .....	18-6
Creating a user-defined pattern .....	18-7
Creating the pattern exemplar (1 of 4) .....	18-9
Creating the pattern exemplar (2 of 4) .....	18-10
Creating the pattern exemplar (3 of 4) .....	18-11
Creating the pattern exemplar (4 of 4) .....	18-12
Pattern Authoring project .....	18-13
Selecting the source files .....	18-14
Configuring the pattern (1 of 6) .....	18-15
Configuring the pattern (2 of 6) .....	18-17
Configuring the pattern (3 of 6) .....	18-18
Configuring the pattern (4 of 6) .....	18-19
Configuring the pattern (5 of 6) .....	18-20
Configuring the pattern (6 of 6) .....	18-22
Setting the Patterns Explorer category .....	18-23
Creating the pattern (1 of 2) .....	18-24
Creating the pattern (2 of 2) .....	18-25
Testing the pattern .....	18-26
Creating the pattern archive .....	18-28
Installing the user-defined pattern .....	18-29
Unit summary .....	18-30
Checkpoint questions .....	18-31
Checkpoint answers .....	18-32
<b>Unit 19. Extending IBM Integration Bus .....</b>	<b>19-1</b>
Unit objectives .....	19-2
IBM Integration Bus EIS adapter nodes .....	19-3
Scenario 1: Legacy data interface to an EIS .....	19-5

Scenario 2: Web services interface to an EIS .....	19-6
Scenario 3: Synchronizing enterprise information systems .....	19-7
Significant extensions for packaged applications .....	19-8
Business activity monitoring with IBM Business Monitor .....	19-9
Monitoring information for IBM Business Monitor .....	19-10
Integration requirements for IBM Business Monitor .....	19-11
Conversion from WebSphere Enterprise Service Bus .....	19-12
Example WebSphere Enterprise Service Bus conversion .....	19-13
Connectivity to IBM Business Process Manager .....	19-14
Business Process Manager Advanced nodes in IBM Integration Bus .....	19-15
Example: Processing an inbound message .....	19-17
Wizards for SCA components .....	19-18
Synergy with Business Process Manager .....	19-19
Flexible development .....	19-20
WebSphere Service Registry and Repository .....	19-21
Integration Bus client for WebSphere Service Registry and Repository .....	19-23
Dynamic runtime selection and invocation .....	19-24
IBM DataPower SOA appliances .....	19-25
DataPower with IBM Integration Bus .....	19-26
WebSphere Transformation Extender .....	19-27
WebSphere Transformation Extender Industry Packs .....	19-28
Sterling Commerce Connect:Direct .....	19-29
Access to mobile services .....	19-30
Integration Bus Fix Pack enhancements (1 of 2) .....	19-31
Integration Bus Fix Pack enhancements (2 of 2) .....	19-32
Unit summary .....	19-33
Checkpoint questions .....	19-34
Checkpoint answers .....	19-35
<b>Unit 20. Course summary .....</b>	<b>20-1</b>
Unit objectives .....	20-2
Course learning objectives .....	20-3
Course learning objectives .....	20-4
Course learning objectives .....	20-5
To learn more on the subject .....	20-6
Unit summary .....	20-7
<b>Appendix A. List of abbreviations.....</b>	<b>A-1</b>

# Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

CICS®	DataPower®	DB™
DB2®	developerWorks®	Express®
IMS™	PartnerWorld®	SurePOS™
Tivoli®	WebSphere®	Worklight®
z/OS®		

Intel and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the “Marks”) of VMware, Inc. in the United States and/or other jurisdictions.

Sterling Commerce® is a trademark or registered trademark of IBM International Group B.V., an IBM Company.

Other product and service names might be trademarks of IBM or other companies.



# Course description

## IBM Integration Bus V10 Application Development II

**Duration:** 4 days

### Purpose

This course provides an intermediate-level continuation of the topics necessary to successfully create IBM Integration Bus message flow applications and integration services. This course focuses on using IBM Integration Bus to develop, deploy, and support platform-independent message flow applications and integration services. These applications and integration services use various messaging topologies to transport data between service requesters and service providers, and also allow the data to be routed, transformed, and enriched during processing. Topics in this course include creating integration services and message flow applications that use and provide web services. You also learn how to use event-driven processing nodes and how to use the record and replay facility to capture and view data during processing. You also learn how IBM Integration Bus interacts with other IBM and enterprise information products. Lab exercises throughout the course give you an opportunity to practice your new skills.

### Audience

This intermediate course is designed for integration specialists and senior-level developers with experience in IBM Integration Bus application development.

### Prerequisites

Before taking this course, you should successfully complete *IBM Integration Bus V10 Application Development I* (WM666G), which introduces IBM Integration Bus development topics that are necessary for success in this course.

### Objectives

After completing this course, you should be able to:

- Use event-driven message processing to control the flow of messages by using message aggregation, message collections, message sequences, and time-sensitive nodes
- Transform data by using Microsoft .NET and XML stylesheets
- Analyze and filter information in complex XML documents

- Extend DFDL message models
- Use message sets and the Message Repository Manager (MRM) parser
- Provide a message flow application as a web service
- Request a web service from within a message flow
- Describe how to implement WS-Addressing and WS-Security standards in IBM Integration Bus
- Create an integration service
- Create and implement an IBM MQ request and response service definition
- Create and implement a database service definition
- Configure security-enabled message processing nodes
- Create a decision service that implements business rules to provide routing, validation, and transformation
- Expose a set of integrations as a RESTful web service
- Use a global cache to store static data
- Record and replay data that a message flow application processes
- Implement publish and subscribe with IBM Integration Bus
- Describe the workload management options for adjusting the message processing speed, and controlling the actions that are taken on unresponsive flows and threads
- Construct user-defined patterns
- Describe how IBM Integration Bus integrates with other IBM products such as IBM WebSphere Enterprise Service Bus and IBM DataPower Appliances

## Curriculum relationship

This course is a follow-on course for WM666, *IBM Integration Bus V10 Development I*. WM676/ZM676 completes some topics that were introduced in WM666, and addresses features of IBM Integration Bus that are not covered in WM666.

# Agenda

## Day 1

- Course introduction
- Unit 1. Using event-driven processing nodes
- Exercise 1. Implementing message aggregation
- Unit 2. Transforming data with Microsoft .NET
- Unit 3. Transforming data with XSL stylesheets
- Unit 4. Analyzing XML documents
- Unit 5. Modeling complex data with DFDL
- Exercise 2. Extending a DFDL model

## Day 2

- Unit 6. Working with message sets and the MRM domain
- Unit 7. Supporting web services
- Exercise 3. Implementing web services
- Unit 8. Developing integration solutions by using integration services
- Exercise 4. Creating an integration service
- Unit 9. Connecting a database by using a discovered service

## Day 3

- Unit 10. Connecting IBM MQ by using a discovered service
- Exercise 5. Creating IBM MQ and database services
- Unit 11. Creating a decision service
- Exercise 6. Creating a decision service
- Unit 12. Developing integration solutions by using a REST API
- Unit 13. Using the global cache
- Unit 14. Implementing message flow security
- Exercise 7. Implementing IBM Integration Bus runtime security

## Day 4

- Unit 15. Implementing publish/subscribe
- Unit 16. Monitoring message flow events
- Exercise 8. Recording and replaying message flow data
- Unit 17. Managing the workload
- Unit 18. Creating patterns for reusability
- Unit 19. Extending IBM Integration Bus
- Unit 20. Course summary



# Unit 1. Using event-driven processing nodes

## What this unit is about

Event-driven message processing nodes control the flow of messages through message flows by using aggregation, message collections, message sequences, and timeout flows. In this unit, you learn how to aggregate and control the sequence of messages in a message flow. You also learn how to use time-sensitive nodes to control when processes run.

## What you should be able to do

After completing this unit, you should be able to:

- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals

## How you will check your progress

- Checkpoint
- Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals

© Copyright IBM Corporation 2016

Figure 1-1. Unit objectives

WM676/ZM6761.0

### Notes:



## How to check online for course material updates



**Note:** If your classroom does not have Internet access, ask your instructor for more information.

### Instructions

1. Enter this URL in your browser:  
[http://www.ibm.com/developerworks/connect/middleware\\_edu](http://www.ibm.com/developerworks/connect/middleware_edu)
2. On the wiki page, locate and click the **Course Information** category.
3. Find your course in the list and then click the link.
4. The wiki page displays information for the course. If an errata document is available, this page is where it is found.
5. If you want to download an attachment, such as an errata document, click the **Attachments** tab at the bottom of the page.  
A screenshot of a wiki page showing a navigation bar with tabs: "Comments (0)", "Versions (1)", "Attachments (1)" (which is highlighted in blue), and "About".
6. To save the file to your computer, click the document link and follow the dialog box prompts.

© Copyright IBM Corporation 2016

Figure 1-2. How to check online for course material updates

WM676/ZM6761.0

### Notes:



## Event-driven processing nodes

- Event-driven processing nodes
  - Aggregate messages
  - Collect messages
  - Sequence and resequence messages
  - Add time-sensitive control
- Require local IBM MQ queue manager that is specified on the integration node
- By default, use SYSTEM.BROKER queues on the integration node queue manager

© Copyright IBM Corporation 2016

Figure 1-3. Event-driven processing nodes

WM676/ZM6761.0

### Notes:

The IBM Integration Toolkit includes some event-driven processing nodes. These nodes are used for message aggregation, timeout, message collections, and message sequences.

Any message flows that contain event-driven processing nodes require that a local IBM MQ queue manager is specified for the integration node.

The event-driven processing nodes use special SYSTEM.BROKER queues, which the queue manager owns, to store information about in-flight messages. The Integration Bus installation includes command files that create the queues.



## Topics

- Message aggregation
- Message collection
- Controlling message sequence
- Timeout nodes

© Copyright IBM Corporation 2016

---

Figure 1-4. Topics

WM676/ZM6761.0

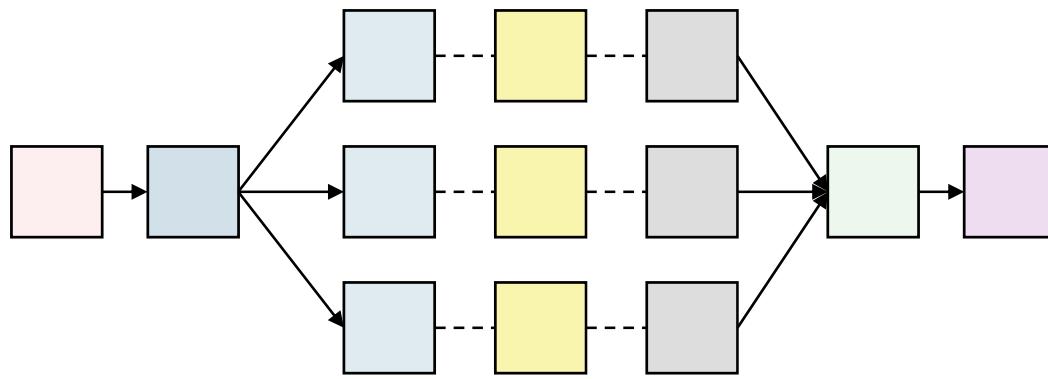
## Notes:



## 1.1. Message aggregation

## Message aggregation

- With message aggregation, you can write a message flow that generates multiple requests from a single input message, and coordinates multiple responses to provide a single consolidated response to that input message
- IBM Integration Bus provides message processing nodes to control the fan-out and fan-in processes to manage the aggregation process



© Copyright IBM Corporation 2016

Figure 1-5. Message aggregation

WM676/ZM6761.0

### Notes:

Message aggregation is an extension of the request/reply processing model. It combines the generation and fan-out of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

Using an aggregation message flow is easier than writing flows with multiple callouts and an MQ Get node (or the equivalent nodes in other message transports) because Integration Bus manages the fan-out and fan-in processing.

## Message aggregation details

- IBM Integration Bus aggregation nodes
  - Aggregate Control node
  - Aggregate Request node
  - Aggregate Reply node
- A message flow that contains the Aggregate Control node followed by an Aggregate Request node initiates message aggregation
- Responses are aggregated by using a flow that contains the Aggregate Reply node
- Aggregation operations are managed with IBM MQ queues
  - By default, uses SYSTEM.BROKER.AGGR.\* queues
  - Control the queues that different aggregation nodes use by creating alternative queues and an **Aggregation** configurable service
- Aggregation nodes are in the **Routing** drawer of the Message Flow editor palette

© Copyright IBM Corporation 2016

Figure 1-6. Message aggregation details

WM676/ZM6761.0

### Notes:

Integration Bus uses an IBM MQ queue manager and special SYSTEM.BROKER.AGGR queues to manage message aggregation operations:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

## Aggregate Control node



- Marks the beginning of a fan-out of requests
- Sends a control message that the Aggregate Reply node uses to match the different requests that are made
- Propagates control information:
  - Integration node identifier
  - **Aggregate Name** property
  - **Timeout** property
- Creates the `LocalEnvironment.ComIbmAggregateControlNode` folder

**Important!** Do not change the contents of this folder

© Copyright IBM Corporation 2016

Figure 1-7. Aggregate Control node

WM676/ZM6761.0

### Notes:

The Aggregate Control node starts an aggregation fan-out. It creates a folder in the `LocalEnvironment` tree that manages the aggregation process.

 **Warning**  
Do not modify the contents of the `LocalEnvironment.ComIbmAggregateControlNode` folder.

The **Control** terminal on the Aggregate Control node is not used. It was deprecated in WebSphere Message Broker V6 and is retained only for compatibility.

## Aggregate Control node configuration



- Set the **Aggregate Name** property of the node to a unique value within the integration node runtime environment that is used for associating the fan-out and fan-in flows
- Set the **Timeout** property to the number of seconds for aggregation to wait for replies to be received during the fan-in
  - 0 (no limit) is the default

© Copyright IBM Corporation 2016

Figure 1-8. Aggregate Control node configuration

WM676/ZM6761.0

### Notes:

The **Aggregate Name** property associates the fan-out and fan-in flows for a specific aggregation flow. Set the **Aggregate Name** property to a value that is unique within the integration node runtime environment. Do not use the same value for this property that is used in another aggregation message flow that runs in the same integration node runtime environment.

The **Timeout** property sets the number of seconds for aggregation to wait for replies from the fan-in. It is a good practice to set the timeout value to a large value; for example, 86400 seconds (24 hours), so that the queues clear old aggregation messages even if timeouts are not required or expected.

## Aggregate Request node



- Records that the request messages were sent
- Collects information that helps the Aggregate Reply node to construct the aggregated reply message
- Adds a folder to the Environment tree that contains control information
  - Set the folder name with the **Folder Name** node property

**Important!** Do not change the contents of this folder

© Copyright IBM Corporation 2016

Figure 1-9. Aggregate Request node

WM676/ZM6761.0

### Notes:

The Aggregate Request node records information about the request messages that are sent to the various services. It stores this information in the Environment tree, in a folder that you specify in the **Folder Name** property of the node.

## Aggregate Reply node



- Marks the end of an aggregation fan-in
- Collects replies and combines them into a single aggregated reply message
- If the Aggregate Reply node does not receive one or more of the response messages, wire nodes to the **Timeout** or **Unknown** terminals to handle the responses that were already received
  - **Unknown**: Terminal to which messages are routed when they cannot be identified as valid reply messages
  - **Timeout**: Terminal to which the incomplete compound message is routed when the timeout that is specified in the corresponding Aggregate Control node expires
- **Aggregate Name** property on the Aggregate Reply node must match the same property on the Aggregate Control node
- **Unknown Message Timeout** property determines how long to hold a message that cannot be identified as part of an aggregation
  - 0 (no limit) is the default

© Copyright IBM Corporation 2016

Figure 1-10. Aggregate Reply node

WM676/ZM6761.0

### Notes:

The Aggregate Reply node ends a fan-in operation.

The Aggregate Reply node **Unknown Message Timeout** property sets the amount of time that “invalid” messages are held until they are propagated to the **Unknown** terminal. This action covers the case in which the first replies arrive, but not all of the requests were sent out.

This action contrasts with the **Timeout** parameter on the Aggregate Control node, which sets the duration to wait for the Aggregate Reply node to receive all aggregation replies. If that **Timeout** property is exceeded, the message assembly is sent to the **Timeout** terminal.

## Implementing message aggregation

1. Wire the aggregation nodes in the message flow.
  - Aggregate Control
  - Aggregate Request
  - Aggregate Reply
2. Define the **Aggregate name**.
  - Unique within all message flows in the integration node
  - Do not deploy the same flow to multiple integration servers in an integration node
  - Multiple instances of the flow are allowed
3. Define a folder name for each request to identify replies in the compound Root tree.
4. Compose a valid message from compound Root tree.
  - ComIbmAggregateReplyBody is not valid for output
5. Plan for error processing.
  - Timeouts
  - Unknown replies

© Copyright IBM Corporation 2016

Figure 1-11. Implementing message aggregation

WM676/ZM6761.0

### Notes:

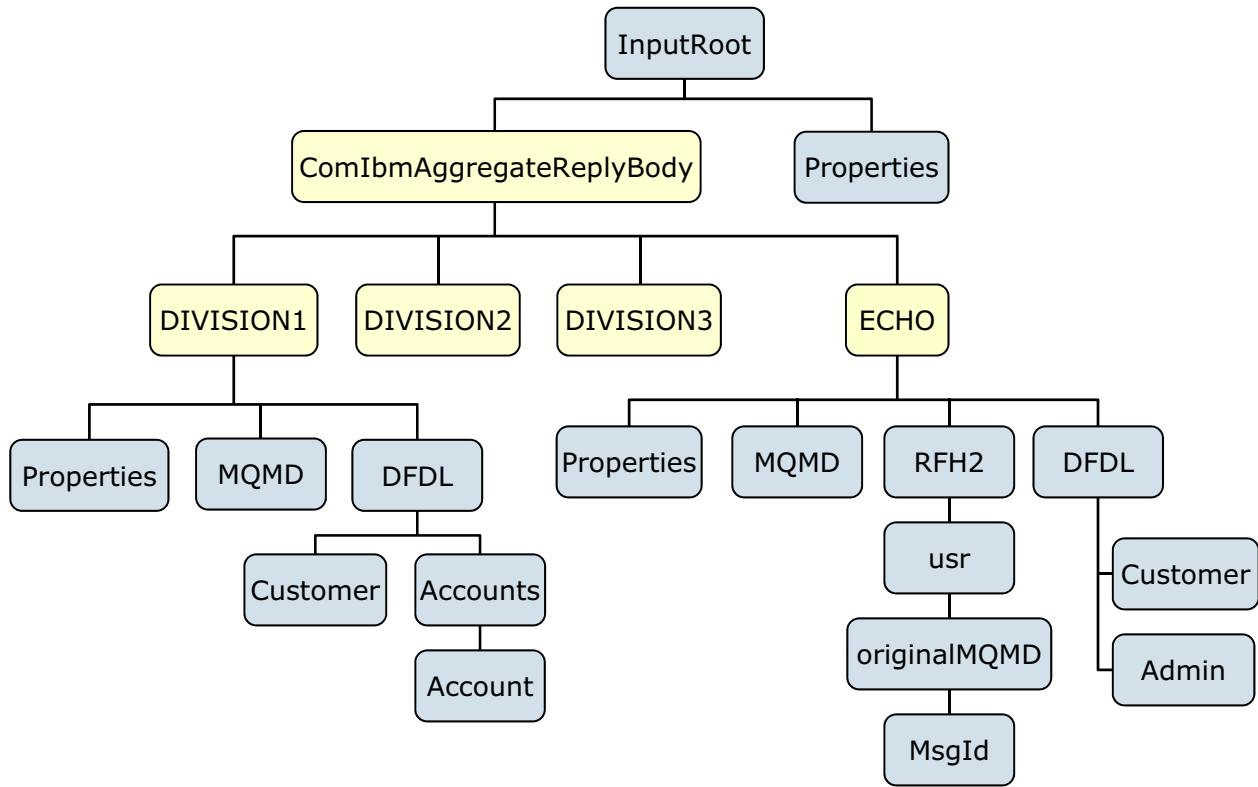
This figure summarizes the steps for implementing message aggregation.

Always plan for error processing by setting timeout values and connecting all output terminals.

In certain situations, it might be necessary to receive an aggregated reply message within a certain time. Some reply messages might be slow to return, or might never arrive. For these situations, set the **Timeout** property of the Aggregate Control node to specify the number of seconds that the integration node must wait for replies. By default, this property is set to 0, which means that no timeout exists, and the integration node waits indefinitely.

If the timeout interval passes without all the replies arriving, the replies that did arrive are turned into an aggregated reply message by the corresponding Aggregate Reply node, and propagated to its **Timeout** terminal. If you choose, you can process this partial response message the same as a complete aggregated reply message. If you prefer, you can provide special processing for incomplete aggregated replies.

## Aggregation message tree



© Copyright IBM Corporation 2016

Figure 1-12. Aggregation message tree

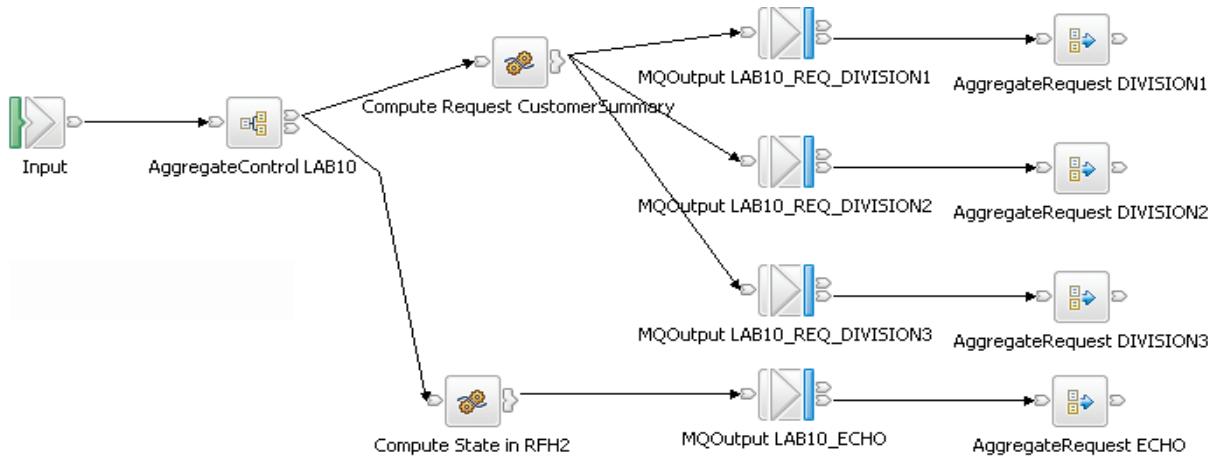
WM676/ZM6761.0

### Notes:

The Aggregate Reply node creates a logical message tree with multiple bodies. Each reply is contained in a separate body folder, which the request name identifies. In the figure, the body folders are DIVISION1, DIVISION2, and DIVISION3.

In a subsequent compute-type node, you must create a valid root message that can be sent to the destination application.

## Message aggregation example: Fan-out flow



Integration node automatically maintains aggregation state in special aggregation queues:

**SYSTEM.BROKER.AGGR.\***

where \* can be:

**Request, Control, Reply, Timeout, Unknown**

© Copyright IBM Corporation 2016

Figure 1-13. Message aggregation example: Fan-out flow

WM676/ZM6761.0

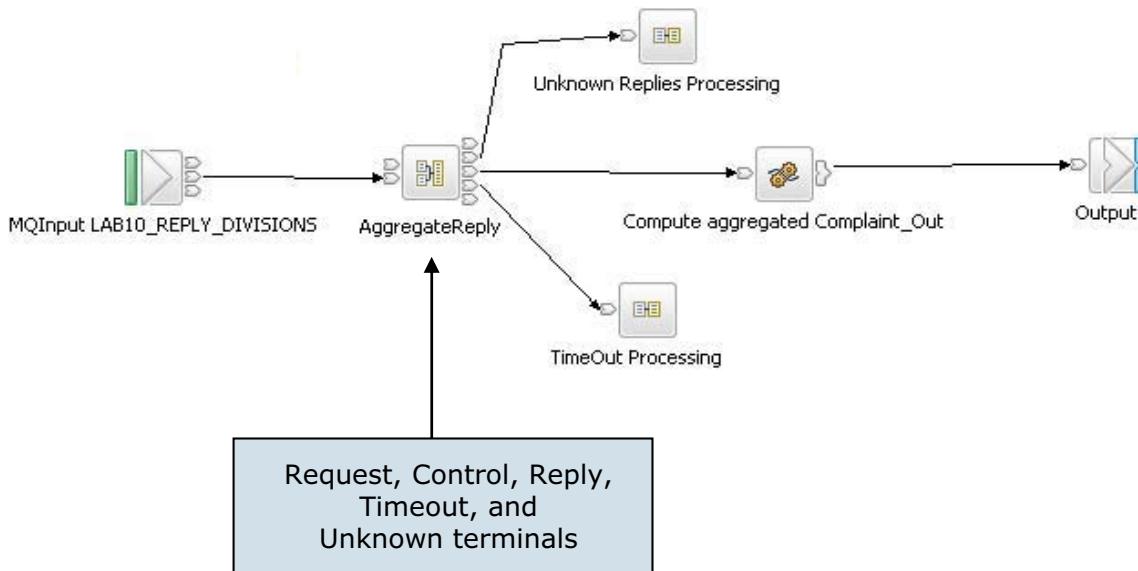
### Notes:

This figure shows an example of a message aggregation fan-out flow.

During the fan-out process, some control information is stored automatically in the `ComIbmAggregateControlNode` folder in the LocalEnvironment tree by the Aggregate Control and Aggregate Request nodes. Additionally, the Aggregate Request nodes store information about awaited requests in the integration node aggregation queues.

After all request messages are sent, the control information is propagated from the Aggregate Control node to the Aggregate Reply node in the fan-in process.

## Message aggregation example: Fan-in flow



© Copyright IBM Corporation 2016

Figure 1-14. Message aggregation example: Fan-in flow

WM676/ZM6761.0

### Notes:

This figure shows an example of a message aggregation fan-out flow.

In the flow, the Aggregate Reply node records any incoming reply from the MQ Input node and stores it until all outstanding replies arrive, or a timeout occurs. Then, the Aggregate Reply node composes a logical message tree with multiple bodies. Each reply is contained in a separate body folder, which the request name identifies.

## Multiple units of work during aggregation

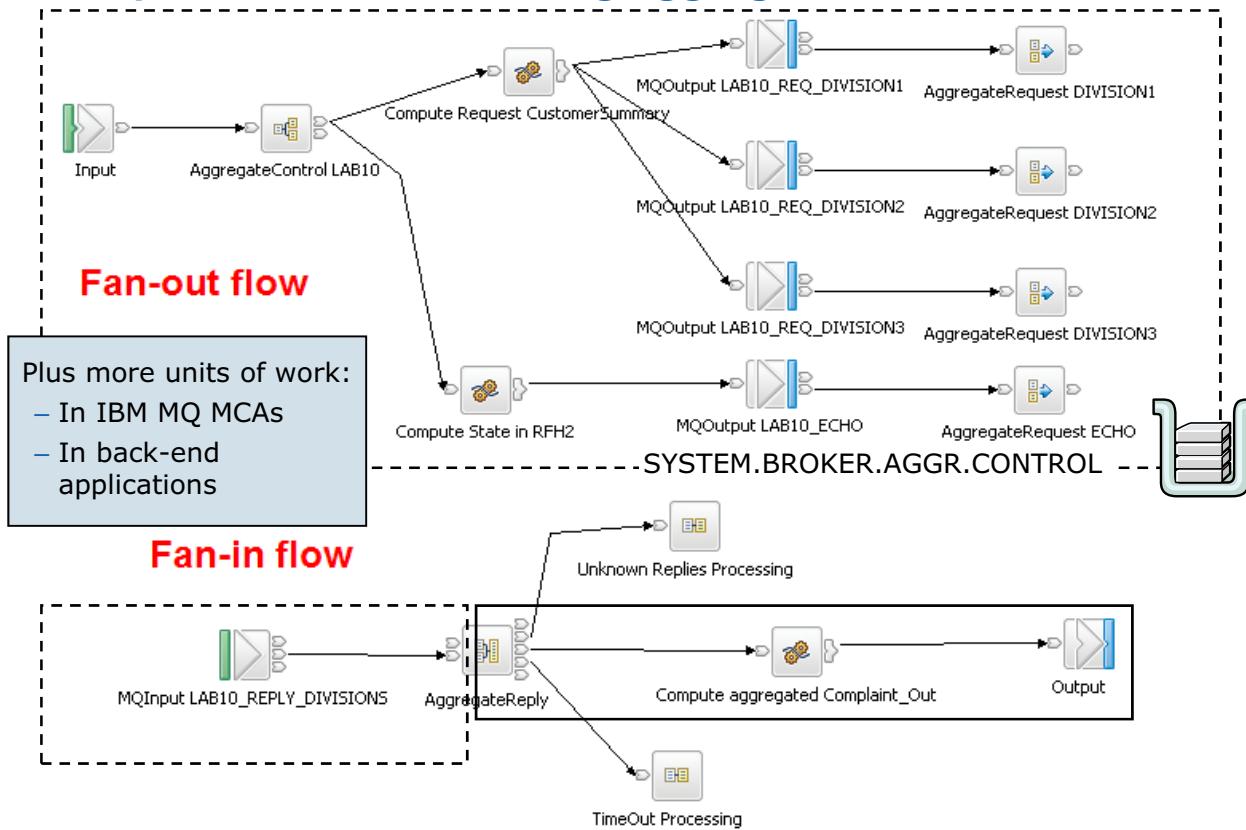


Figure 1-15. Multiple units of work during aggregation

WM676/ZM6761.0

### Notes:

An aggregation flow is a complex application that involves multiple transactions (units of work).

1. In a fan-out flow, the first logical unit of work ends when all requests are sent, and the automatic control message is put onto the **SYSTEM.BROKER.AGGR.CONTROL** queue.
2. In a fan-in flow, each incoming reply has a logical unit of work while aggregating from the MQ Input node to the Aggregate Reply node.
3. For the aggregated result message, a new logical unit of work starts in the Aggregate Reply node (which has a **TransactionMode** property) when all the replies arrive, or the timeout interval expires.

These logical units of work are the only ones in a message flow. There are more logical units of work during IBM MQ transport, and in those applications that use the requests and send replies to the fan-in flow.

Logical units of work are important for error handling and recovery. Be sure to consider connecting **Catch** terminals to capture any errors during message processing.

## Aggregation implementation tips (1 of 2)

- Unknown replies cannot occur if:
  - Fan-out transactional
  - Good design and security prevent “rogue” messages
- Add flow instances to increase throughput
  - Typically more instances for fan-in flow
  - Deploy fan-in flow only in one integration server to centralize timeout processing
- Implement fan-out and fan-in as separate flows for more flexibility

© Copyright IBM Corporation 2016

Figure 1-16. Aggregation implementation tips (1 of 2)

WM676/ZM6761.0

### Notes:

When a message arrives at the In terminal of an Aggregate Reply node, it is examined to see whether it is an expected reply message. If it is not recognized, it is propagated to the **Unknown** terminal. You might want the integration node to wait for a specific duration before propagating the message because the reply message might arrive before the work that the Aggregate Request node does is transactionally committed.

By putting the fan-out flow under transactional control, you ensure that the fan-out flow completes before any response messages get to the Aggregate Reply. If the unknown timeout interval expires and the message is recognized, it is processed. The node also checks to see whether this previously unknown message is the last reply that is needed to make an aggregation complete. If it is, the aggregated reply message is constructed and propagated. If the unknown timeout interval expires and the message is still not recognized, the message is propagated to the unknown terminal.

You can create the fan-out and fan-in flows, either in the same message flow or in two different message flows. In either case, the two parts of the aggregation are associated by setting the **Aggregate Name** property. A single flow is easier to implement for a simple case, but this approach has some limitations, and you might find that you prefer the flexibility of two message flows.

- The two flows can be modified independently of each other.
- The two flows can be stopped and started independently of each other.
- The two flows can be deployed to separate integration servers to take advantage of multiprocessor systems, or to provide data segregation for security or integrity purposes.

## Aggregation implementation tips (2 of 2)

- You can use more than one Aggregate Control node in a message flow
  - Use the same value for **Aggregate Name** property in all, but use different values for the **Timeout** property
  - This case is the only time where you can reuse an aggregate name

### Use case:

If you create an aggregation flow that books a business trip, you might have some requests that need a reply within two days. However, more urgent requests need a reply within 2 hours.

© Copyright IBM Corporation 2016

Figure 1-17. Aggregation implementation tips (2 of 2)

WM676/ZM6761.0

### Notes:

You can use more than one Aggregate Control node in a message flow to provide more flexibility.



## 1.2. Message collection

## Message collection

- Single message that contains multiple messages that are derived from one or more sources so that downstream nodes can process them together
  - Use a Collector node to group messages from one or more sources into a message collection
  - Manually build a message collection by using a Compute node
- Relies on IBM MQ queues for the storage of in-flight messages
  - By default, uses SYSTEM.BROKER.EDA.\* queues
  - Control the queues that different Collector nodes use by creating alternative queues, and a **Collector** configurable service
- Only the following nodes can process message collections:
  - Compute
  - JavaCompute
  - .NETCompute

© Copyright IBM Corporation 2016

Figure 1-18. Message collection

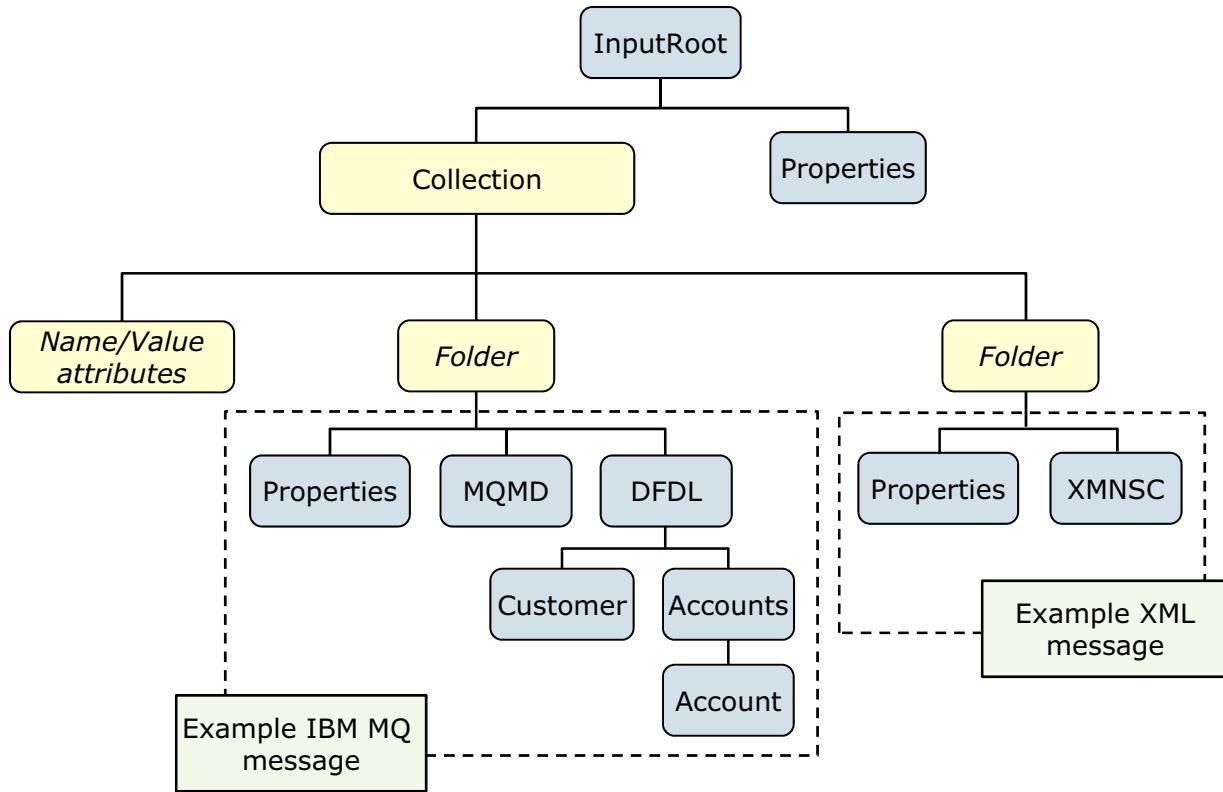
WM676/ZM6761.0

### Notes:

A message collection is a message that contains multiple messages that are derived from one or more sources. Message collections can be created in a message flow by using a Collector node.

Compute, JavaCompute, and .NETCompute nodes can process message collections.

## Message collection message tree



© Copyright IBM Corporation 2016

Figure 1-19. Message collection message tree

WM676/ZM6761.0

### Notes:

This figure shows an example of the logical message tree that the Collector node creates. The message tree in this example contains two messages, one received from IBM MQ, and one from a file input source.

A message collection has a **Properties** header and a single folder element named **Collection**. A message collection can also have zero or more attributes that are name-value pairs; the name of an attribute must be unique within a message collection. These pairs are shown as *Name/Value* in the figure.

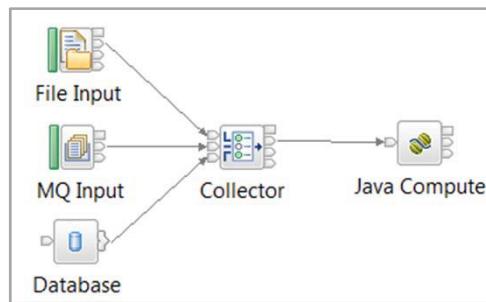
A standard attribute for the message collection is an attribute that is called *CollectionName*. If you use a Collector node to generate a message collection, the value for the collection name is generated based on the values you configure in the node. The collection name attribute is not compulsory.

The Collection folder in the message collection tree structure contains other folders, which are shown as *Folder* in the example. These folders contain the message tree of each message that is added to the message collection. Each of these folders has a name, but this name does not have to be unique within the message collection. The value for the *Folder* name is derived from the source

of the input message. You can use ESQL or XPath expressions to access the content of messages in a message collection by referencing the folder names that `InputRoot.Collection` precedes.

## Collector node

- Creates message collections based on rules that you configure in the node
  - Dynamic input terminals can receive messages from different sources
  - Event handlers determine how messages are added to a message collection, and when a message collection is complete
- Options for event coordination processing
  - Collections are propagated when complete
  - Collections are held until message is received on Control terminal
- Set an expiry timeout for message collections that fail to be completed in a satisfactory time



© Copyright IBM Corporation 2016

Figure 1-20. Collector node

WM676/ZM6761.0

### Notes:

The Collector node creates message collections that are based on rules that you define on the node.

You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, which are known as *event handlers*, to determine how messages are added to a message collection, and when a message collection is complete.



## 1.3. Controlling message sequence

## The message sequencing problem

Problem:

Messages must be processed in a specific order to maintain the integrity of a workflow

Solution:

Two nodes that remove the need for custom logic and state management

- Sequence node stamps a sequence number on a message
- Resequence node stores and forwards in strict sequence order

© Copyright IBM Corporation 2016

Figure 1-21. The message sequencing problem

WM676/ZM6761.0

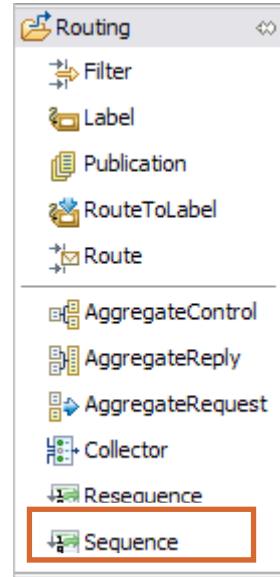
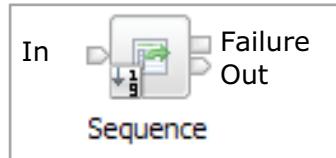
### Notes:

Often, messages must be processed in a specific order. For example, a series of debits and credits against a bank account must be processed in the order they took place. Patient records that are received, processed, and forwarded must be sent in the order in which they arrived.

Integration Bus contains two nodes, Sequence and Resequence to solve this sequencing problem.



## Sequence node



- Allocates a sequence number to each received message
- Propagates a message that is stamped with the sequence number
- Multiple “sequence groups” are handled independently, in parallel
- Does not allocate the next sequence number in a group until the current message in group finishes processing to allow sequencing across multiple threads
- Sequence group state is preserved across integration node restarts

© Copyright IBM Corporation 2016

Figure 1-22. Sequence node

WM676/ZM6761.0

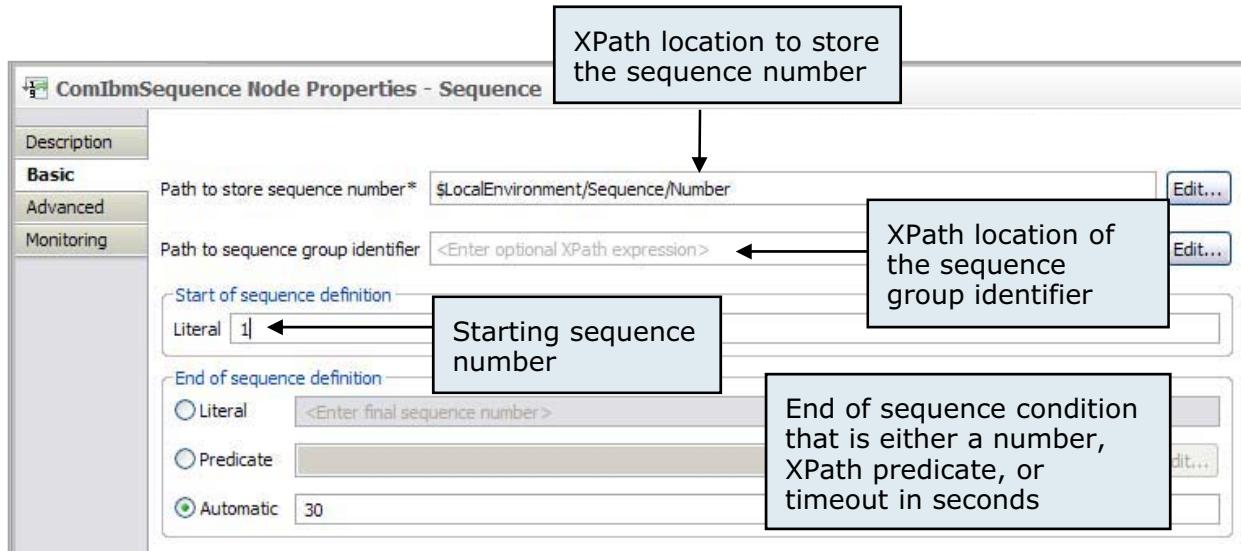
### Notes:

The Sequence node receives an input message, allocates a monotonically increasing sequence number from a user-defined start point, and stores the number in a user-defined location in the message assembly. It does this process for each message until the sequence ends, which a user-defined condition determines.

Messages that are arriving can be divided into independent sequence groups that are based on an identifier in the message. Each group has its own sequence number and is handled independently. The SYSTEM.BROKER.SEQ.GROUP and SYSTEM.BROKER.SEQ.NUMBER queues store the current sequence number for each active sequence group. Sequence numbers can be persisted so that the sequence survives across queue manager restarts. The node ensures that the next sequence number for a group is not allocated until the current message for that group is either committed or rolled back.

If another thread is processing a message in the same group, the thread attempts to obtain the IBM MQ state message for the group. This process ensures that sequencing is maintained for the group when it has multiple threads.

## Sequence node properties



© Copyright IBM Corporation 2016

Figure 1-23. Sequence node properties

WM676/ZM6761.0

### Notes:

The figure shows an example of the Sequence node properties.

A literal number must specify the start of the sequence, but the local environment `LocalEnvironment.Sequence.StartOfSequenceNumber` can override it.

A literal number, an XPath predicate that evaluates to true or false, such as `$Body/doc/last=true()`, or the time to wait for the next message identifies the end of the sequence.

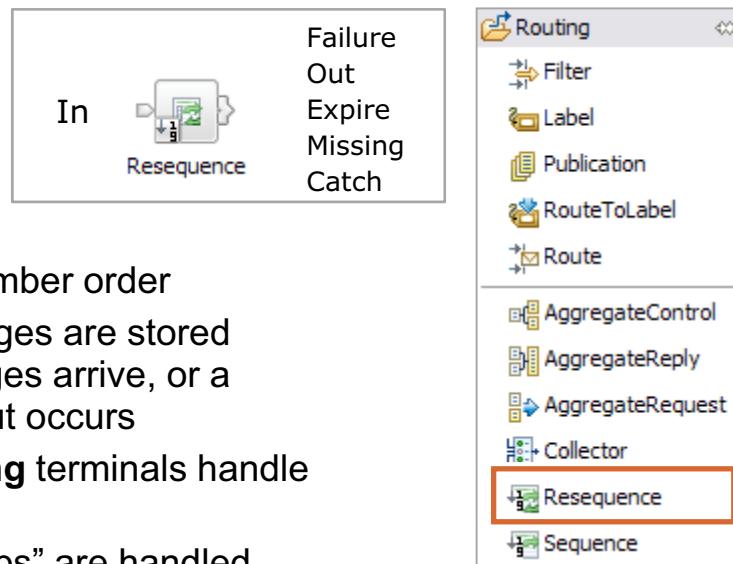
In addition to the location that was given to store sequence number, the sequence number is also set in the LocalEnvironment, with the sequence group identifier and the start and end of sequence flags:

- `LocalEnvironment.Sequence.Number`
- `LocalEnvironment.Sequence.Group`
- `LocalEnvironment.Sequence.Start`
- `LocalEnvironment.Sequence.End`

The **Advanced** tab properties control persistence and specify a configurable service.

## Resequence node

- Examines a sequence number in each received message
- Propagates messages only in the sequence number order
- Out-of-sequence messages are stored until the missing messages arrive, or a missing message timeout occurs
- **Out, Expire, and Missing** terminals handle different use cases
- Multiple “sequence groups” are handled independently, in parallel
- Sequence group state is preserved across integration node restarts
- Node is a transaction break (like the Collector node)
- Contains a retry mechanism to control the handling of downstream failures



© Copyright IBM Corporation 2016

Figure 1-24. Resequence node

WM676/ZM6761.0

### Notes:

The Resequence node receives an input message and checks a sequence number in a user-defined location in the message assembly. If it is the next message in the sequence, it propagates the message to the **Out** terminal.

If an out-of-sequence message arrives, the node stores the message until the missing messages arrive. A missing message timeout can be specified to prevent the node from waiting indefinitely for missing messages. When the timer expires, all stored messages are propagated in order, one at a time, to the **Expire** terminal. If the missing messages eventually arrive, they are propagated to the **Missing** terminal. Any other messages that arrive are propagated to the **Expire** terminal.

After a sequence starts, as determined by a user-defined condition, resequence processing continues for each message until the sequence ends, which a user-defined condition also determines.

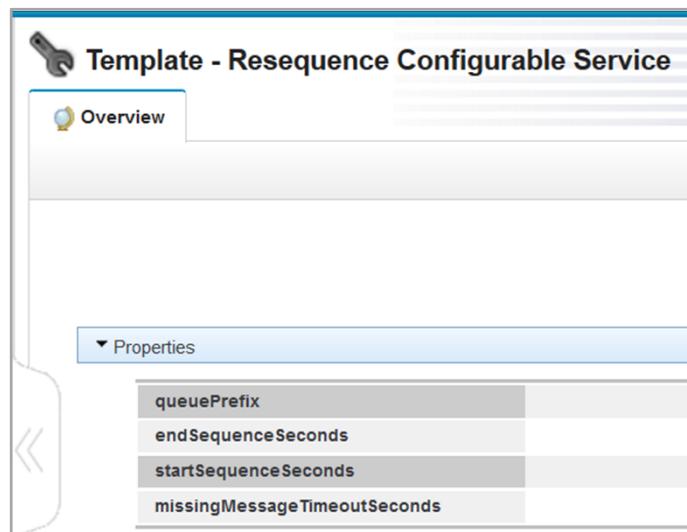
Similar to the Sequence node, messages can be divided into independent sequence groups. Each group has its own sequence number and is handled independently, enabling the parallel processing of groups.

The **Retry mechanism** property specifies how the Resequence node handles a flow failure.

- If the **Failure** option is selected and a message fails to propagate to the **Out** and **Catch** terminals, the message is propagated to the **Failure** terminal. The Resequence node is returned to a state where it is waiting for that failing message, which its sequence number identifies, to arrive again. Any subsequent messages in the sequence that are already received are not sent out until the message that failed is re-sent to the node. Any subsequent messages in the sequence that arrive are held as usual.
- If the **Infinite** option is selected, any exceptions that occur in nodes that follow the Resequence node are rolled back to the **Catch** terminal of the Resequence node. If the **Catch** terminal is not connected to any other nodes, the messages are redelivered to the original terminal. The messages are never backed out or discarded.

## Resequence node storage queues

- By default, the storage queues used by all Resequence nodes are:
  - SYSTEM.BROKER.EDA.EVENTS
  - SYSTEM.BROKER.EDA.COLLECTIONS
  
- Control the queues different Resequence nodes
  1. Create alternative queues.
  2. Create a **Resequence** configurable service.
  3. Specify the **Configurable service** property on the Resequence node.



© Copyright IBM Corporation 2016

Figure 1-25. Resequence node storage queues

WM676/ZM6761.0

### Notes:

Information about the state of in-flight messages is held on storage queues that the IBM MQ queue manager controls.

If the integration node has the necessary permissions to create the default system queues, they are created automatically when a flow that contains Resequence nodes is deployed. If the default queues are not created automatically, you can create them manually.

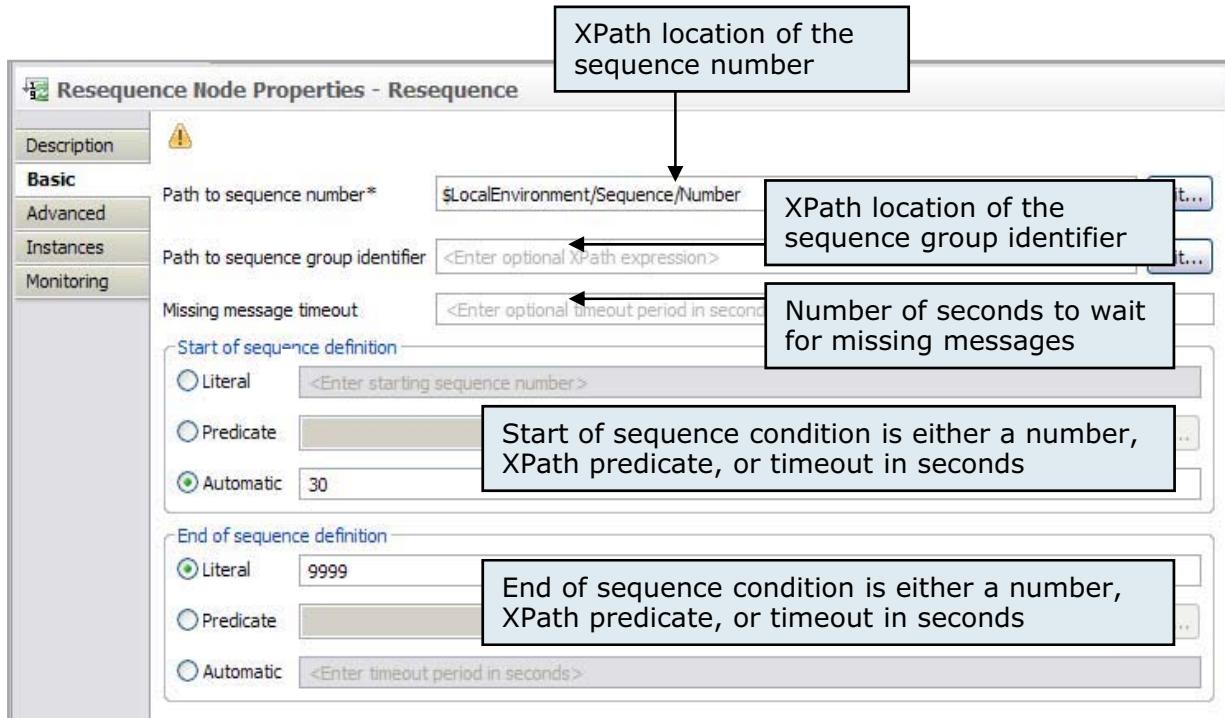
By default, the storage queues used by all Resequence nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

You can control the queues that are used by different Resequence nodes by creating alternative queues by using a Resequence configurable service to specify the names of those queues for storing events.

You can control the queues that are used by different Resequence nodes by creating alternative queues by using a Resequence configurable service to specify the names of those queues for storing events. For more information, see the “Configuring the storage of events for Resequence nodes” topic in the IBM Knowledge Center for IBM Integration Bus.

## Resequence node properties



© Copyright IBM Corporation 2016

Figure 1-26. Resequence node properties

WM676/ZM6761.0

### Notes:

The figure shows the Resequence node **Basic** properties. A literal number specifies the start of the sequence, an XPath predicate that evaluates to true or false, such as \$Body/doc/first=true(), or a time to wait. If a wait time is specified, the node stores all messages until the timer expires. Then, it uses the lowest number as the start of the sequence.

A literal number specifies the end of the sequence, an XPath predicate that evaluates to true or false, or the time to wait for the next message.

The sequence number is copied to the local environment, with the sequence group identifier and the start and end flags:

- LocalEnvironment.Sequence.Number
- LocalEnvironment.Sequence.Group
- LocalEnvironment.Sequence.Start
- LocalEnvironment.Sequence.End

If a missing message creates a gap between the previously propagated message and the current message, the sequence numbers of the missing messages are stored in the LocalEnvironment.Sequence.Missing location with one entry for each missing number.

The **Instances** tab allows the Resequence node to request extra threads from the message flow's thread pool, or to allocate a separate thread pool for its own use.

The **Advanced** tab properties control persistence and specify a configurable service.

## Resequence node use cases

- Case 1: Messages must never go out of sequence
  - Wire the **Out** terminal to the main-line flow
  - Wire the **Expire** and **Missing** terminals to separate branches for requeuing
- Case 2: Missing messages can be tolerated, but all other messages must remain in sequence
  - Wire the **Out** and **Expire** terminals to the main flow path
  - Leave the **Missing** terminal unwired (to discard) or wire to a separate branch
- Case 3: Some out-of-sequence messages can be tolerated
  - Wire the **Out**, **Expire**, and **Missing** terminals to the main flow path

© Copyright IBM Corporation 2016

Figure 1-27. Resequence node use cases

WM676/ZM6761.0

### Notes:

This figure lists some examples of when the Resequence nodes would be used:

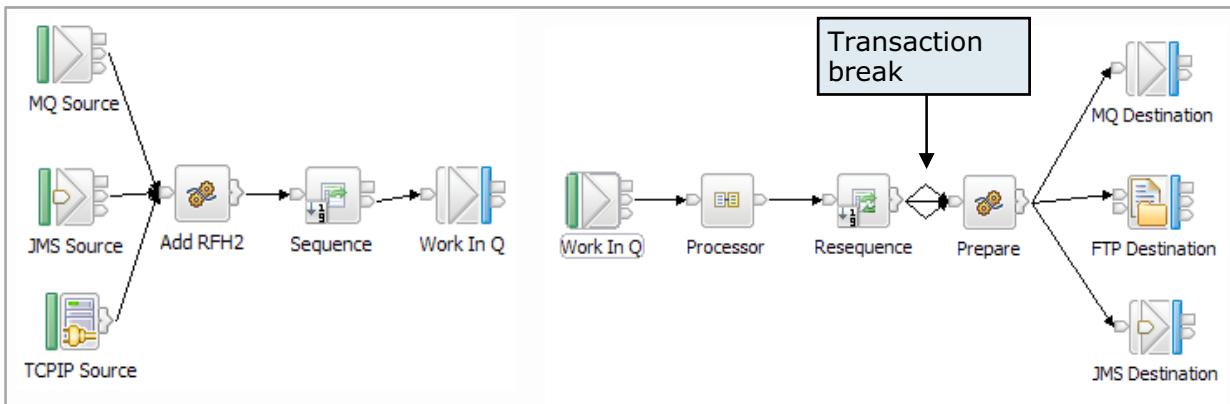
**Case 1:** Messages must never go out of sequence. If a message is missing, route all subsequent messages (in sequence, if possible) to a temporary queue after a timeout period. To configure this behavior, wire the **Out** terminal to the main flow and the **Expire** and **Missing** terminals to separate branches (most likely for requeuing).

**Case 2:** Missing messages can be tolerated, but all other messages must remain in sequence. If a message is missing, skip over it and continue processing the rest of the sequence. If the missing message eventually arrives, either discard it, or process it separately from the main processing. To configure this behavior, wire the **Out** and **Expire** terminals to the main flow and leave the **Missing** terminal unwired (to discard the message). You can also wire the **Missing** terminal to a separate branch.

**Case 3:** Some out-of-sequence messages can be tolerated.

The flow must process all of the messages, preferably in sequential order, but some messages can be skipped and processed later to not delay the flow for too long. To configure this behavior, wire the **Out**, **Expire**, and **Missing** terminals into the main flow.

## Using the nodes as a pair



- Scenario: Messages without a sequence number arrive from multiple sources
  - Message order must be preserved when they are sent to their destinations
  - Intermediate processing depends on the size and type of message and must be highly parallel to achieve required throughput
- Solution: Use a sequence group for each source
  - Use a receiver flow with a Sequence node to establish the sequence
  - Add a Resequence node to the processing flow to preserve the sequence
  - Ensure that sequence start and end conditions agree

© Copyright IBM Corporation 2016

Figure 1-28. Using the nodes as a pair

WM676/ZM6761.0

### Notes:

The message flows in the figure show how the sequence and resequence nodes can be used together. In this example, the Sequence node establishes the sequence that is based on the order of arrival. The Resequence node reestablishes the sequence in case it is out of step due to the length of time that was taken during the main processing phase of the flow (which is shown as a subflow).

In the figure, the Resequence node **Out**, **Expire**, and **Missing** terminals are all wired to the next node in the flow for convenience only.

The Sequence and Resequence nodes must detect the start and end of the sequence together. One way to synchronize the nodes is for the Resequence node to use the `LocalEnvironment.Sequence.Start` and `LocalEnvironment.Sequence.End` flags that the Sequence node sets.

The Resequence node is a transaction break. It takes three units of work to process a message in this scenario. You can use separate thread pools for the left side and right side of the processing flow to enable the necessary throughput without starving the right side.



## 1.4. Timeout nodes

## Timeout nodes

- Allow the integration node to start a flow from a timed event
  - Timeout Notification starts the flow (mandatory)
  - Timeout Control generates a configuration message for the Timeout Notification node (optional)
- Flexible timer characteristics
  - Particular times
  - Regular intervals
  - Multiple simultaneous requests
  - Many timer controls can point to a single timer notification
- Configurable
  - Node properties
  - Configuration message

© Copyright IBM Corporation 2016

Figure 1-29. Timeout nodes

WM676/ZM6761.0

### Notes:

You can use the time-sensitive nodes to run processes at specific times, or at fixed intervals, and act when transactions are not completed within a defined time.

For example, you might want to have a certain flow run hourly at the top of the hour by using a UNIX `cron` job. This job would presumably use a utility, such as `amqspput`, to send a message to a queue that would be the input queue of the message flow you would like to run.

While this approach certainly works, it adds overall complexity to the system. You must coordinate the creation of the `cron` job with the deployment of the message flows. If it has many flows, then maintenance of the `cron` becomes burdensome. Another point of failure is introduced into the system and you must have a mechanism for ensuring that the `amqspput` worked and did not get a bad return code, such as 2035. This approach makes problem determination more complex.

Using timeout nodes provides a flexible, self-contained mechanism for starting a message flow. No external coordination (to the integration node) is required beyond what is normally required to deploy a flow. Built-in integration node trace mechanisms can be used to detect the failure to set the timers.

## Storage queues for Timeout nodes

- By default, the storage queue that is used by all Timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE
- You can control the queue that is used by different Timeout nodes by creating an alternative queue and by using a Timer configurable service to specify the name of that queue

© Copyright IBM Corporation 2016

Figure 1-30. Storage queues for Timeout nodes

WM676/ZM6761.0

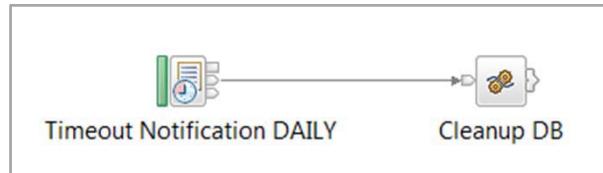
### Notes:



## Stand-alone (automatic) Timeout Notification

- Triggers a message flow at regular time intervals
- Timer starts with a flow start (deploy)
- The message consists of Root.Properties only
  - Plus LocalEnvironment.TimeoutRequest
- Usage patterns

Example: Maintenance flows to clear  
the database one time per day



Timeout Notification Node Properties - Timeout Notification DAILY	
Description	
Basic	Unique identifier* DAILY
Parser Options	Transaction mode* Yes
Validation	
Monitoring	Operation mode* Automatic
	Timeout interval (sec) 1440

© Copyright IBM Corporation 2016

Figure 1-31. Stand-alone (automatic) Timeout Notification

WM676/ZM6761.0

### Notes:

The TimeoutNotification node would be the first node in a timer controlled message flow. It has no In terminal.

You can use TimeoutNotification nodes by themselves or coupled with one or more TimeoutControl nodes.

As a stand-alone node, TimeoutNotification nodes support simple, fixed interval timer events. For example, you can have a message flow that runs every 30 seconds, or every hour, or every 17 years.

When only a TimeoutNotification node is used in a message flow, it has little flexibility. The node begins timing as soon as the message flow is deployed. The first timer event occurs as soon as the flow is deployed. The timer cannot be stopped (except by stopping the message flow), nor is there any way to dynamically change the time interval. Also, if a TimeoutNotification node specifies an event to occur every hour, the event happens one time an hour after the deployment occurred. So, if the flow was deployed at 6:17 AM, then events occur at 6:17 AM, 7:17 AM, 8:17 AM, and every hour thereafter.

This stand-alone scenario would most likely occur if you needed an integration node flow to run for a maintenance purpose, such as processing or clearing rows that accumulated in a database table. Generally, if these maintenance procedures occur on a periodic basis (for example, clear the database one time every hour), it is not important exactly when they occur.

## TimeoutRequest message

- To set a controlled timeout, send a message with a set of elements with well known names to a Timeout Control node
  - Specified in the Timeout Control node input message or in the LocalEnvironment tree
  - Message format (XML is shown, but all parsers are supported)
  - Schema definition is included with the Integration Bus

```
<TimeoutRequest>
  <Action> SET | CANCEL</Action>
  <Identifier> String (arbitrary, alphanum)</Identifier>
  <StartDate> String (TODAY | yyyy-mm-dd)</StartDate>
  <StartTime> String (NOW | hh:mm:ss)</StartTime>
  <Interval> Integer (seconds)</Interval>
  <Count> Integer (greater than zero or -1)</Count>
  <IgnoreMissed> TRUE | FALSE</IgnoreMissed>
  <AllowOverwrite> TRUE | FALSE</AllowOverwrite>
</TimeoutRequest>
```

© Copyright IBM Corporation 2016

Figure 1-32. TimeoutRequest message

WM676/ZM6761.0

### Notes:

The purpose of a TimeoutControl node is to allow for setting the parameters of a TimeoutNotification node. By including one or more TimeoutControl nodes in either the same or different message flows, you can increase the flexibility of the patterns of timer events that you can generate.

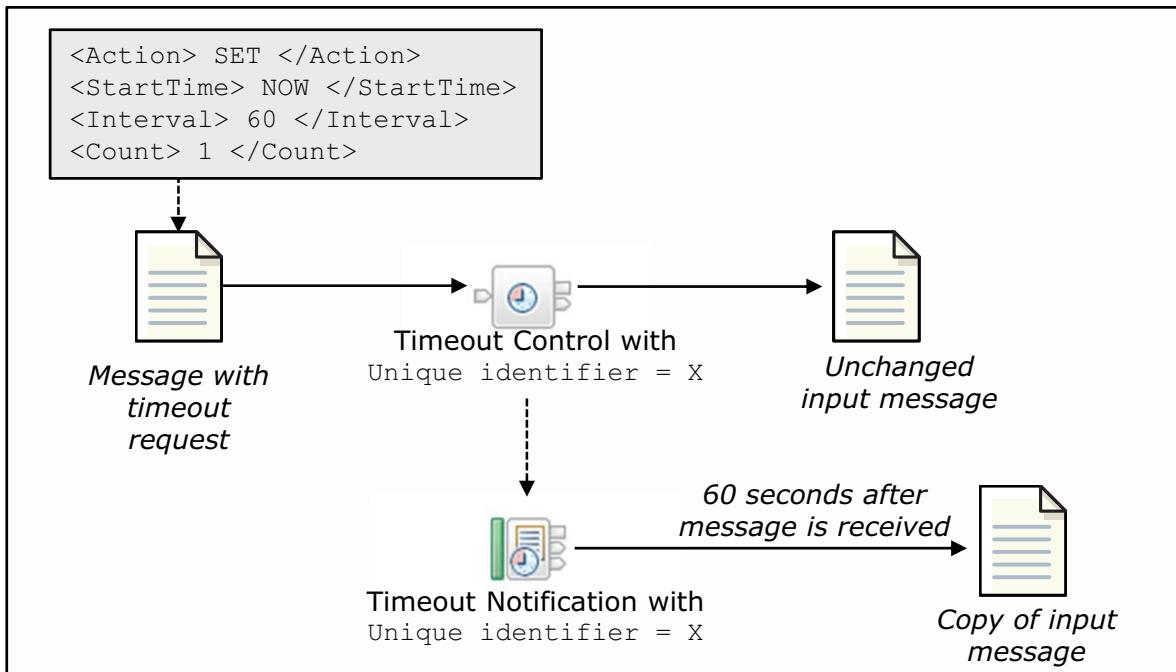
The TimeoutControl node receives an input message that contains a timeout request. The node validates the request, stores the message, and propagates the message (unchanged) to the next node in the message flow.

All Timer nodes create or use a special timeout request structure. You can build the TimeoutRequest structure in the message flow. Any parser can be used for this task, but a predefined schema definition of the timeout request is provided in the IBM Integration Toolkit.

The stand-alone (**Automatic mode**) TimeoutNotification node produces content similar to the following and propagates it in LocalEnvironment (shown as output from a Trace node).

```
(0x01000000):TimeoutRequest = (
(0x03000000):Action      = 'SET'
(0x03000000):Identifier   = 'DAILY'
(0x03000000):StartDate    = '2015-08-13'
(0x03000000):StartTime    = '17:34:53.794321'
(0x03000000):Count        = 1
(0x03000000):Interval     = 1440
(0x03000000):IgnoreMissed = TRUE
(0x03000000):AllowOverwrite = TRUE
```

## Timeout nodes example



Use Timeout Control and Timeout Notification nodes to send a message into a message flow 60 seconds after the message is received

© Copyright IBM Corporation 2016

Figure 1-33. Timeout nodes example

WM676/ZM6761.0

### Notes:

The example shows the path of a message that contains a timeout request through a Timeout Control node. A Timeout Notification node with an identifier that matches the Timeout Control node then processes the timeout request. The example also shows the message that the Timeout Notification node produces after processing the timeout request.

The message comes into the Timeout Control node with the following values set in the TimeoutRequest section of the message:

```

<Action> SET </Action>
<StartTime> NOW </StartTime>
<Interval> 60 </Interval>
<Count> 1 </Count>
    
```

The Timeout Control node validates the TimeoutRequest; default values are assumed for properties that are not explicitly defined. The original message is then sent on to the next node in the message flow.

A property of the Timeout Notification node is its **Unique Identifier**, which is a 1 – 12-character label. If the TimeoutRequest is valid, the Timeout Notification node with the same **Unique identifier**

as the Timeout Control node propagates a copy of the message to the message flow 60 seconds after the message was received.

The **Unique Identifier** must be globally unique within the integration node instance in which this node is deployed. Multiple Timeout Control nodes can control the same Timeout Notification node by specifying the same **Unique Identifier**.

## Unit summary

Having completed this unit, you should be able to:

- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals

© Copyright IBM Corporation 2016

---

Figure 1-34. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: If an Aggregate Reply node receives a message that does not belong to an aggregation flow, the message flow stops abnormally.
  
2. True or false: You can safely deploy different message flows across different integration servers that use the same **Aggregation Name** property value.

© Copyright IBM Corporation 2016

Figure 1-35. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

1.

2.



## Checkpoint answers

1. True or false: If an Aggregate Reply node receives a message that does not belong to an aggregation flow, the message flow stops abnormally.

Answer: **False**. The message flow does not stop when it receives a message that does not belong to the aggregation flow.

2. True or false: You can safely deploy different message flows across different integration servers that use the same **Aggregation Name** property value.

Answer: **False**. You must deploy message flows that use the same Aggregation Name to the same integration server.

© Copyright IBM Corporation 2016

Figure 1-36. Checkpoint answers

WM676/ZM6761.0

### Notes:

## Exercise 1



Implementing message aggregation

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 1-37. Exercise 1

WM676/ZM6761.0

### Notes:

In this exercise, you implement a message aggregation by creating a flow that requests customer information from a back-end system and aggregates the replies.



## Exercise objectives

After completing this exercise, you should be able to:

- Add IBM Integration Bus SYSTEM.BROKER queues to a queue manager
- Use the Aggregate Control node and the Aggregate Request node to generate and concurrently fan-out related requests
- Use the Aggregate Reply node to aggregate messages into a single output message

© Copyright IBM Corporation 2016

Figure 1-38. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.

# Unit 2. Transforming data with Microsoft .NET

## What this unit is about

You can use .NET applications on Windows integration nodes to create, route, and transform data by using the .NETCompute and .NETInput nodes. This unit describes the .NETCompute node and the .NETInput node.

## What you should be able to do

After completing this unit, you should be able to:

- Use .NET applications on Windows integration nodes to create, route, and transform messages

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center for IBM Integration Bus



## Unit objectives

After completing this unit, you should be able to:

- Use .NET applications on Windows integration nodes to create, route, and transform messages

© Copyright IBM Corporation 2016

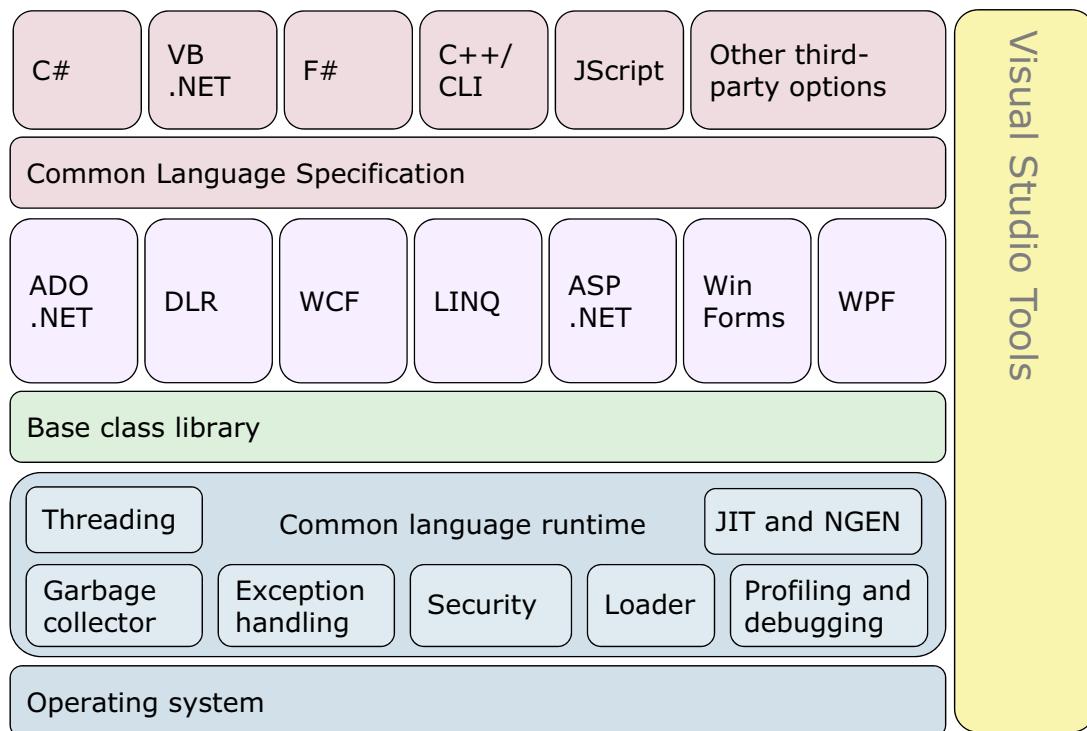
Figure 2-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Microsoft .NET framework overview



© Copyright IBM Corporation 2016

Figure 2-2. Microsoft .NET framework overview

WM676/ZM6761.0

### Notes:

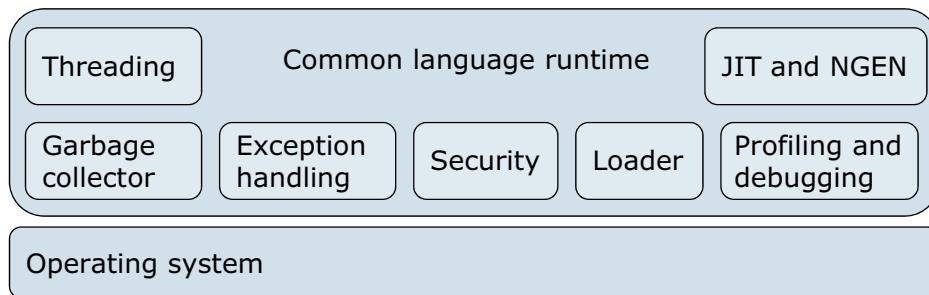
The Microsoft .NET framework is a software framework that runs primarily on Microsoft Windows. It includes a large library and provides language interoperability across several programming languages. Programs that are written for the .NET framework are run in a software environment that is known as the common language runtime (CLR). CLR is an application virtual machine that provides services such as security, memory management, and exception handling.

The base class library of the .NET framework provides the user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Programmers produce software by combining their own source code with the .NET framework and other libraries.

Visual Studio is the Microsoft integrated development environment generally for .NET.

## .NET common language runtime (CLR)

- Provides an environment inside which the “managed” code is run
- Can be hosted inside another process to run the “managed” code
- Provides key services to all the code that runs inside it



© Copyright IBM Corporation 2016

Figure 2-3. .NET common language runtime (CLR)

WM676/ZM6761.0

### Notes:

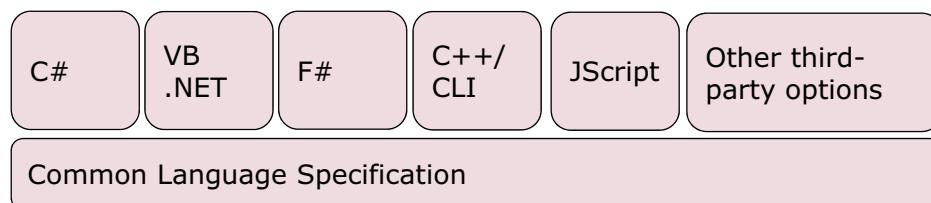
The CLR runs the code and provides services that make the development process easier.

Compilers and tools provide the CLR functions so that you can write code that uses this managed runtime environment.

CLR provides key services such as cross-language integration, cross-language exception handling, enhanced security, deployment support, a simplified model for component interaction, and debugging and profiling services.

## From source code to byte code

- All .NET code is compiled from the source language into managed “CIL” (MSIL) code
- Common Type System (CTS) and Common Language Specification (CLS)
- CIL code is in a .dll or .exe and is called an *assembly*
  - Assembly is loaded into the CLR to be run
  - Code is created just before it is run
- At run time, the CLR does not care what the source language was



© Copyright IBM Corporation 2016

Figure 2-4. From source code to byte code

WM676/ZM6761.0

### Notes:

Common Intermediate Language (CIL) is the lowest-level human-readable programming language that the common language infrastructure (CLI) specification defines. Languages that target a CLI-compatible runtime environment compile to CIL, which is assembled into an object code that has a bytecode-style format. CIL is an object-oriented assembly language, and is entirely stack-based. Its bytecode is translated into operating system code or run by a virtual machine.

A Common Language Specification (CLS) is a document that defines how computer programs can be turned into bytecode. The .NET Framework uses a CLS so that it can interact with other objects regardless of the language.

The .NETCompute node uses any language that complies with CLR to create or modify the message, such as C#, Visual Basic (VB), F#, and C++/CLI.



## Running .NET in your enterprise

- Microsoft .NET is supported on Windows
- .NET function is available in all editions of IBM Integration Bus
- Use .NET applications on Windows integration nodes to create, route, and transform messages
- Tightly integrated with Visual Studio
  - IBM Integration Toolkit can start Visual Studio
  - Drag-and-drop from Visual Studio plug-in and Windows file system simplifies node configuration

© Copyright IBM Corporation 2016

Figure 2-5. Running .NET in your enterprise

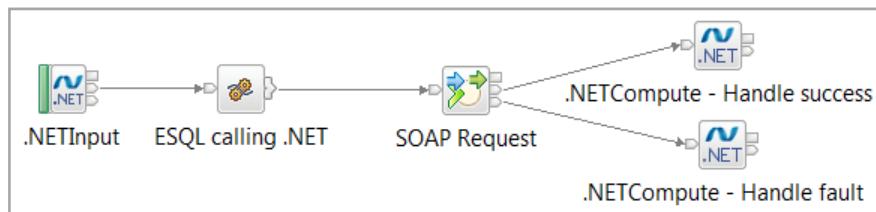
WM676/ZM6761.0

### Notes:

The Microsoft .NET Framework runs on Windows. The .NETCompute node runs only on Windows integration nodes.

You can use .NET applications on Windows integration nodes to create, route, and transform messages by using the .NETCompute and .NETInput nodes. With the .NETInput node, you can position .NET connectivity at the beginning of the message flow. With the .NETCompute node, you can position .NET connectivity in the middle of the message flow as a part of your core infrastructure.

## Integrating .NET with IBM Integration Bus



- Use the .NETInput node to get data from disparate sources such as MSMQ, IBM MQ, file system, or database
- Use the .NETCompute node to construct output messages and interact with Microsoft .NET Framework or Component Object Model (COM) applications
- Use the CREATE FUNCTION or CREATE PROCEDURE statements in a node that supports ESQL to call .NET code

© Copyright IBM Corporation 2016

Figure 2-6. Integrating .NET with IBM Integration Bus

WM676/ZM6761.0

### Notes:

With Integration Bus, you can use the Microsoft .NET and CLR environments to transform messages in a message flow.

You can call .NET assemblies in your message flows from the .NETCompute node, .NETInput node, or from an ESQL procedure. You can also include .NET assemblies as dependencies of a library. These assemblies run in a .NET application domain and can be packaged in a BAR file. Application domains are shown in the navigator view as peers of applications and libraries.

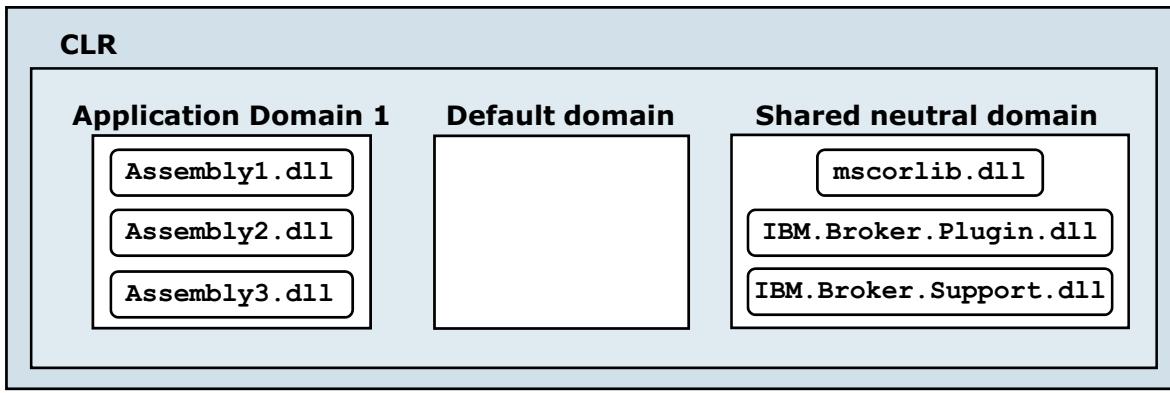
Integration Bus can integrate with Microsoft Visual Studio 2010 or 2012 to create a .NET assembly that represents the node.

To start Microsoft Visual Studio directly from the node, ensure that Microsoft Visual Studio 2010 or 2012 is installed, and double-click the .NETCompute node. Alternatively, right-click the node and click **Open Microsoft Visual Studio**. If a Microsoft Visual Studio project or solution name is provided on the Visual Studio window, the project or solution automatically loads when Microsoft Visual Studio is opened.

## Scope of .NET code in IBM Integration Bus

- Integration server hosts the Microsoft CLR
- All code that CLR runs is in an *application domain*
- Application domains can be created for user code and provide a scope for the code at run time
- Code in other application domains can share code in the *shared neutral domain*

### IBM Integration Bus integration server



© Copyright IBM Corporation 2016

Figure 2-7. Scope of .NET code in IBM Integration Bus

WM676/ZM6761.0

### Notes:

At run time, an integration server hosts the Microsoft CLR.

You configure a .NET node with a .NET assembly that contains the code of the node. The code consists of a class that is derived from the `NBComputeNode` abstract class that is provided in the `IBM.Broker.Plugin.dll` assembly.

The .NET assembly is run inside a .NET application domain. A .NET application domain is a runtime container for .NET assemblies and associated resources that the .NET code in the message flows uses.

If you do not explicitly create a .NET application domain, and do not set a value for the `AppDomain` property on the .NET node, .NET assemblies can also be contained in an implicit .NET application domain.

## .NET application domains in IBM Integration Bus

- Provide the unit of scoping for all .NET code
- Have several properties
  - A name and a “base directory” for the application domain code
  - An optional configuration file that provides extra information to code running in that domain
- Can create by name
  - If unnamed, a domain is named after the IBM Integration Bus application that contains the message flow
- Can also create with a configurable service
  - Specify application domain properties
  - If the code is changed, flow can dynamically reload associated application domains
  - Application domains provide statistics about their memory usage

© Copyright IBM Corporation 2016

Figure 2-8. .NET application domains in IBM Integration Bus

WM676/ZM6761.0

### Notes:

A named .NET application domain can be associated with IBM Integration Bus applications. Assemblies, or associated resources, contained in the .NET application domain are deployed with the referencing message flow or application.

Integration Bus has an XML configuration file that is used to configure the domain. It is used to set container and application-specific properties and is loaded when the application domain is initialized. The value can specify a base name or a fully qualified path. The integration node looks up a base name in the directory that is specified in the **Application Base** property.

The DotNetAppDomain configurable service can be used to specify application domain properties.



## .NETCompute node

- Write your transformations in any CLR-compliant language
  - Build transformations in C#, VB, F#, C++/CLI, or JScript
- Integrate your .NET code directly with your message flow
  - To configure the node, drag an assembly file from a file explorer to the node
- Implements a single method “Evaluate”
- Provides full access to the IBM Integration Bus message trees
- Dynamically defined output terminals

© Copyright IBM Corporation 2016

Figure 2-9. .NETCompute node

WM676/ZM6761.0

### Notes:

The .NETCompute node is similar to the JavaCompute node, in that you can use it to create, modify, or filter a message.

The .NETCompute node uses any CLR-compliant language to create or modify the message, such as C#, Visual Basic, F#, and C++/CLI.

The .NETCompute node class (`NBComputeNode`) contains a method that must be overridden, and two more optional methods that you can choose to override if necessary. You must always override the `Evaluate()` method. If the Integration Bus template is used to generate the skeleton code, the template automatically implements the method.

You can customize the node to create an output message, or a number of messages, by using an input message or data from an external source.

The .NETCompute node has one output terminal but you can define more dynamic output terminals as required.



## .NETCompute node configuration

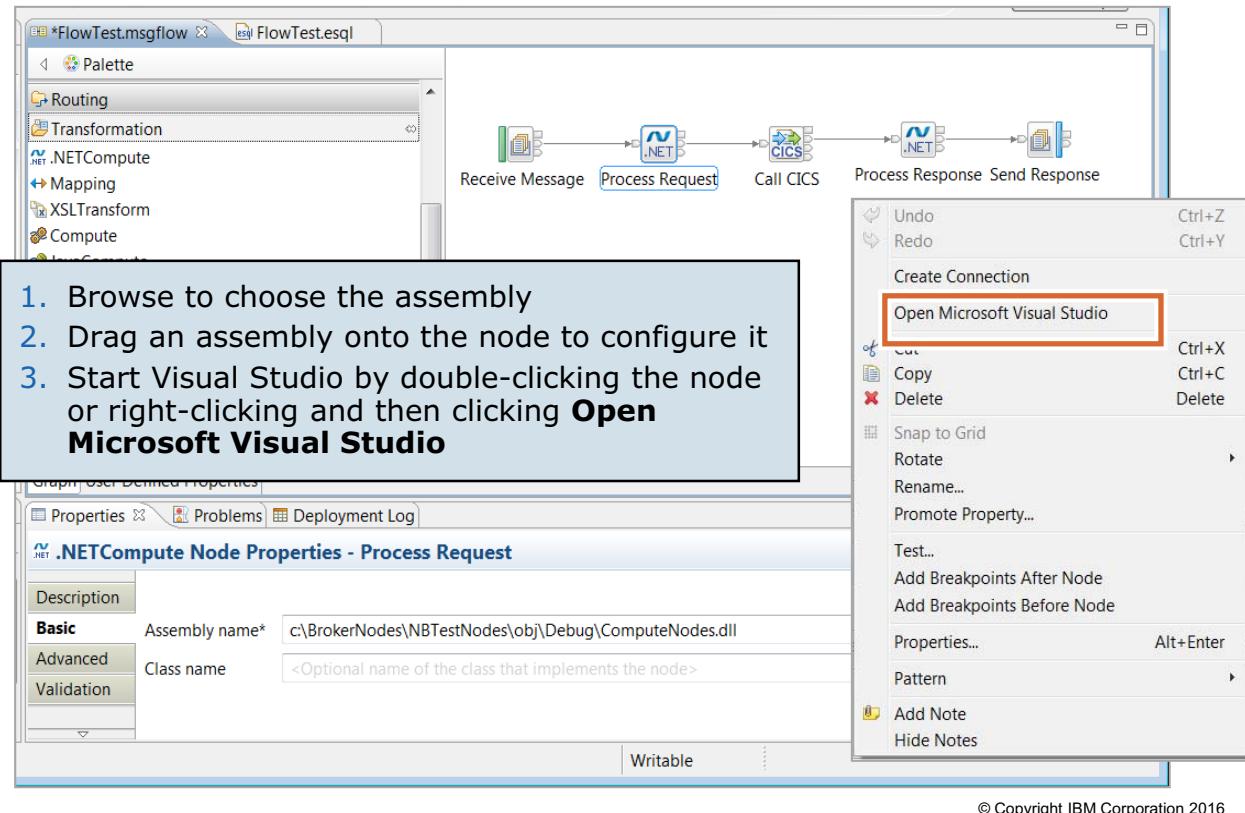


Figure 2-10. .NETCompute node configuration

WM676/ZM6761.0

### Notes:

You create a .NETCompute message flow by using both the IBM Integration Toolkit and Microsoft Visual Studio. You can use any CLR-compliant language to code the .NETCompute node.

If you have a vendor-supplied application .dll file, you must set the .NETCompute node **Assembly name** property and other node properties. If you do not have a .dll, you can create one by using Microsoft Visual Studio. Visual Studio provides development templates for common message flow operations, such as creating a message, modifying a message, or filtering a message based on its content.

After you create the .NET assembly, create a .NETCompute node in the Integration Toolkit. Drag the .NET assembly file (from Windows Explorer) onto the .NETCompute node, and the Integration Toolkit automatically sets several of the properties.

Microsoft Visual Studio must be installed on the same workstation where the Integration Toolkit is installed to start Visual Studio from the .NETCompute node.

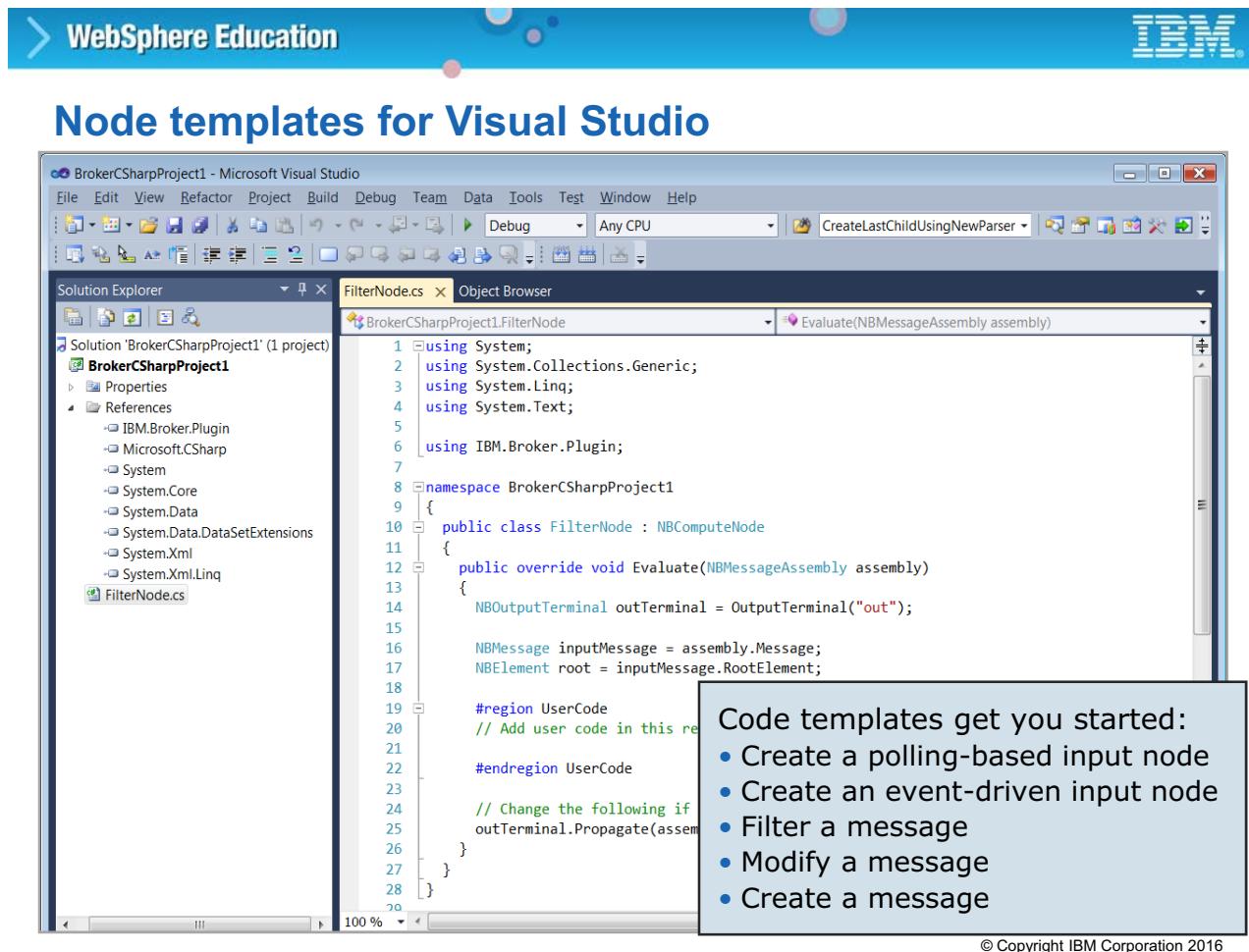


Figure 2-11. Node templates for Visual Studio

WM676/ZM6761.0

## Notes:

Integration Bus can integrate with Microsoft Visual Studio 2010 or 2012 to create a .NET assembly that represents the node.

If the Integration Toolkit is installed after Microsoft Visual Studio, the Project templates with skeleton code are automatically installed ready for you to use.

For the .NETCompute node, you can choose to:

- Create an IBM Integration message
- Filter an IBM Integration message
- Modify an IBM Integration message

For the .NETInput node, you can choose to:

- Create a polling-based IBM Integration input node
- Create an event-driven IBM Integration input node

If the Integration Toolkit is installed first, you must manually install the Microsoft Visual Studio templates. This installation can be achieved by running the file `IBM.Broker.DotNet.vsix` and stepping through the wizard, accepting the license file as part of the process.



## Debug .NET code with the Visual Studio Debugger

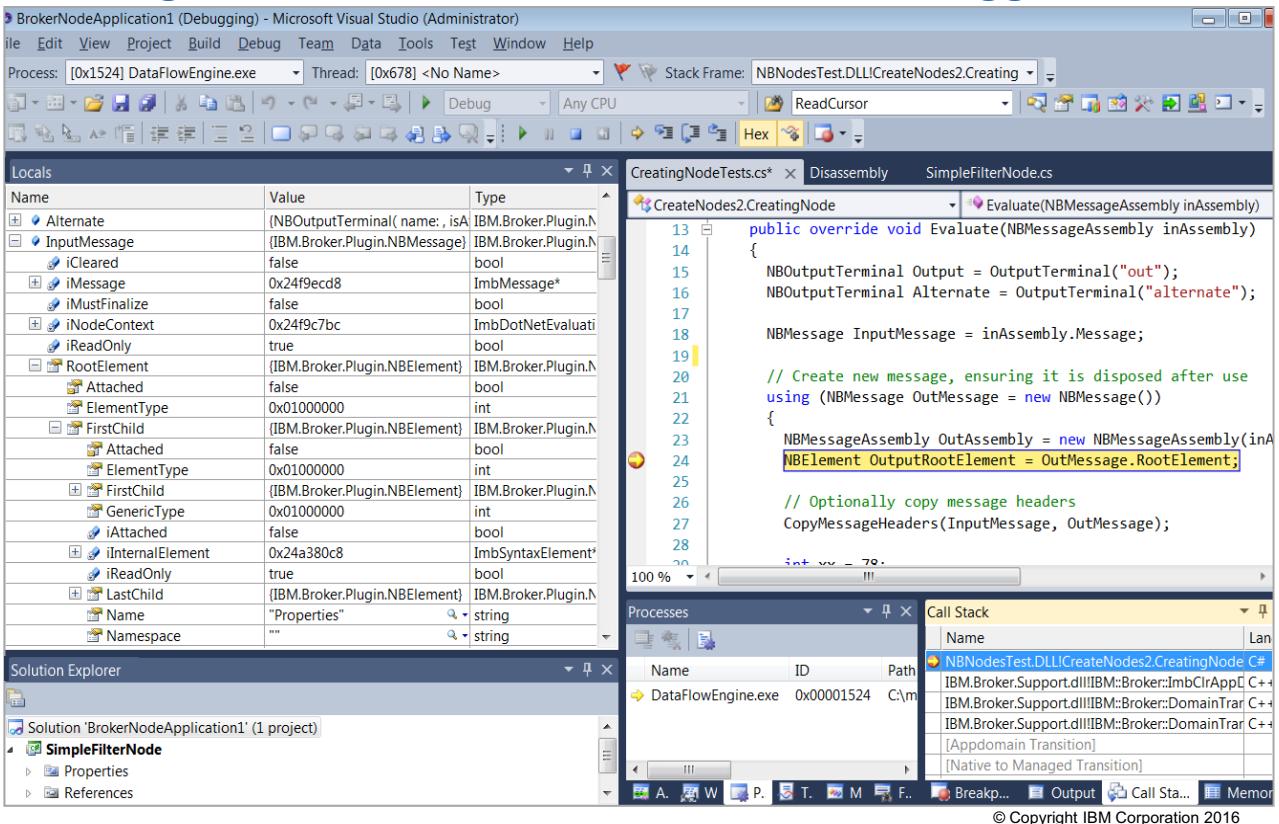


Figure 2-12. Debug .NET code with the Visual Studio Debugger

WM676/ZM6761.0

### Notes:

To debug .NET code that is running inside the integration server:

1. Put the `.pdb` file for the assembly that implements the node in the same directory as the assembly for the node.
2. Attach Microsoft Visual Studio to the `DataFlowEngine.exe` process.
3. Set breakpoints in your code and examine variables: for example, when the breakpoint is reached.

## Handling exceptions

- IBM Integration Bus exceptions are transformed into NBExceptions, which can be detected in .NET code
- If generated from user .NET code, NBExceptions are transformed into IBM Integration Bus exceptions
  - Exit the .NET code with an exception if necessary
  - Detect the exception with a Try node or Catch node, or by wiring a **Catch** terminal
  - NBRecoverable exceptions can be detected in an ESQL handler, with a specified SQLCode and SQLState

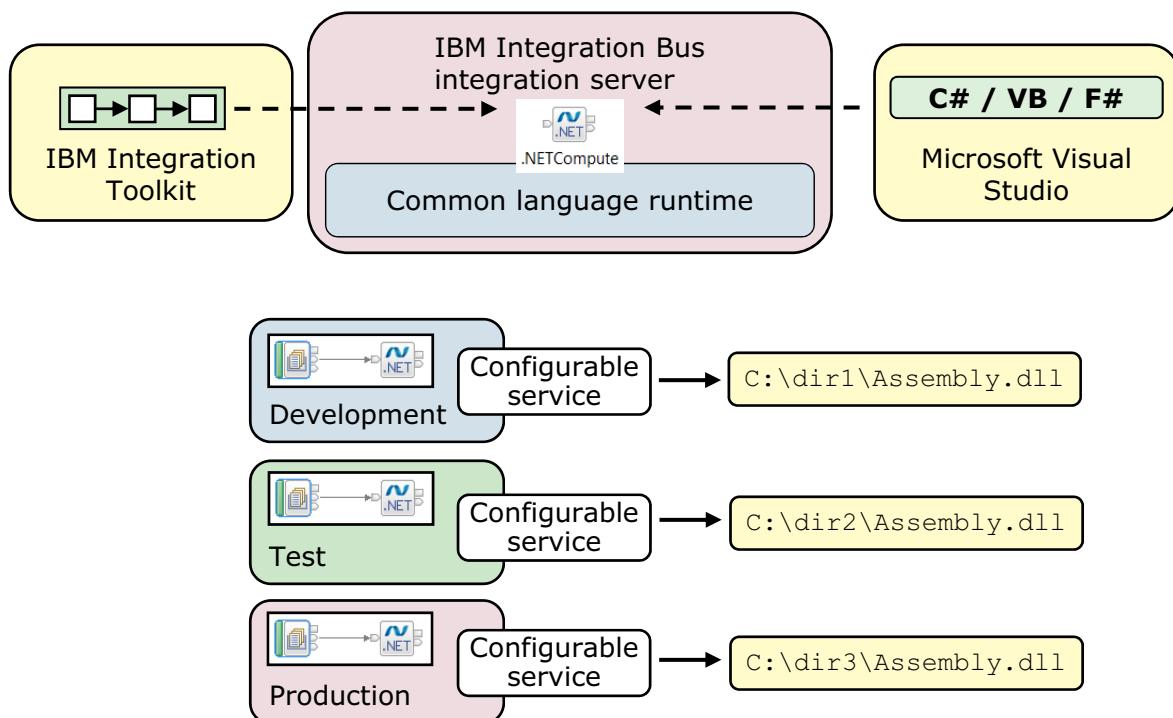
### Notes:

Any Integration Bus exceptions that occur at run time generate NBExceptions. NBException represents the base class of the integration node exception hierarchy from which all integration node exceptions are derived.

In the .NET code, any NBExceptions are transformed into Integration Bus exceptions.

As with any other message processing node, you can optionally wire the **Failure** terminal and the **Catch** terminal of the input node to handle the message.

## Configurable services for .NET (1 of 2)



© Copyright IBM Corporation 2016

Figure 2-14. Configurable services for .NET (1 of 2)

WM676/ZM6761.0

### Notes:

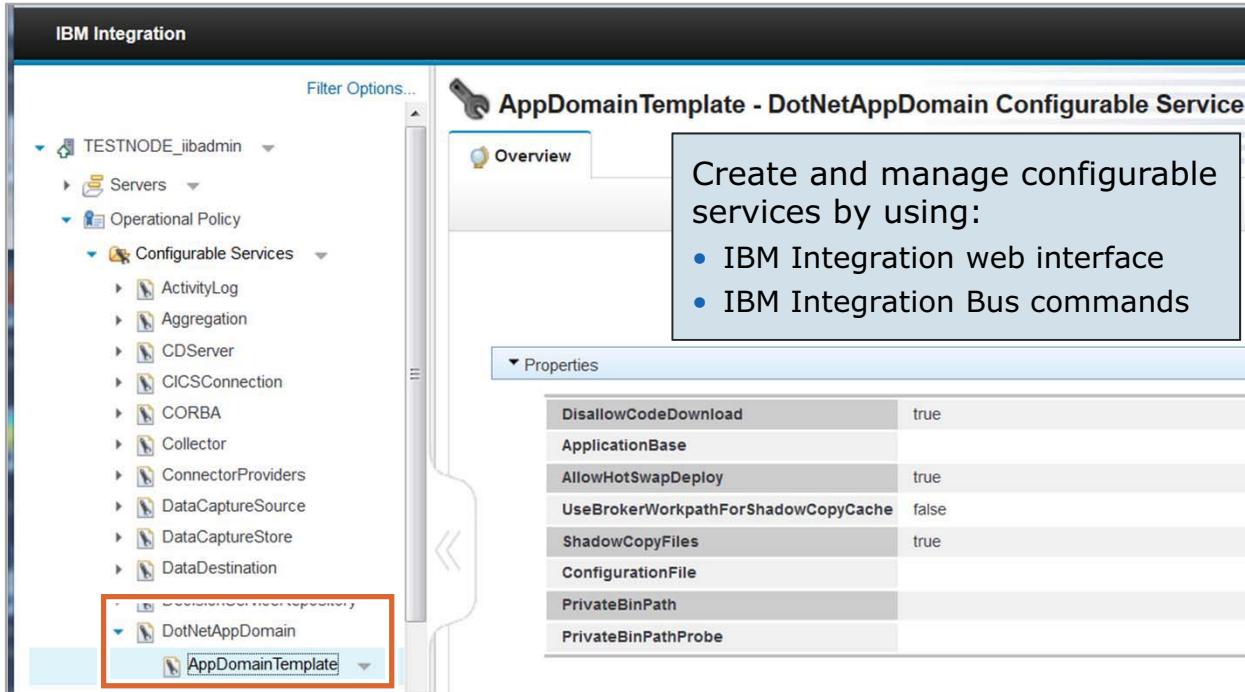
The properties on the .NET node or on the ESQL Procedure signature control the .NET configuration for an integration node. Optionally, a DotNetAppDomain configurable service can also be defined for further configuration options.

You can create a DotNetAppDomain configurable service to connect to .NET. At run time, you can reference a different assembly for the development, test, and production environments, as shown in the figure.

You must stop and start the integration server for a change in a property value to take effect in the DotNetappDomain configurable services.

 WebSphere Education 

## Configurable services for .NET (2 of 2)



The screenshot shows the IBM Integration web interface. On the left, there is a navigation tree under 'TESTNODE\_ibadmin' with nodes like 'Servers', 'Operational Policy', and 'Configurable Services'. Under 'Configurable Services', several items are listed: 'ActivityLog', 'Aggregation', 'CDServer', 'CICSConnection', 'CORBA', 'Collector', 'ConnectorProviders', 'DataCaptureSource', 'DataCaptureStore', 'DataDestination', 'DotNetAppDomain' (which is highlighted with a red box), and 'AppDomainTemplate' (which is also highlighted with a red box). The main panel shows the 'AppDomainTemplate - DotNetAppDomain Configurable Service' details. It has tabs for 'Overview' (selected) and 'Properties'. A callout box points to the 'Properties' tab with the text: 'Create and manage configurable services by using: • IBM Integration web interface • IBM Integration Bus commands'. Below the tabs, there is a table of properties:

DisallowCodeDownload	true
ApplicationBase	
AllowHotSwapDeploy	true
UseBrokerWorkpathForShadowCopyCache	false
ShadowCopyFiles	true
ConfigurationFile	
PrivateBinPath	
PrivateBinPathProbe	

© Copyright IBM Corporation 2016

Figure 2-15. Configurable services for .NET (2 of 2)

WM676/ZM6761.0

### Notes:

You can create and manage the .NET configurable services in the IBM Integration web interface or by using the `mqsicreateconfigurableservice` IBM Integration Bus command.

The properties for the DotNetAppDomain configurable service are:

- **DisallowCodeDownload** controls whether the .NET Application Domain is allowed to download dependent code from outside the integration node server.
- **ApplicationBase** is the base directory for the application domain from which assemblies are loaded.
- **AllowHotSwapDeploy** allows the application domain to be reloaded by changing the assemblies on disk below the ApplicationBase without requiring the flow to be redeployed.
- **UseBrokerWorkpathForShadowCopyCache** controls the location that is used for shadow copying of assemblies.
- **ShadowCopyFiles** controls whether assemblies for this application domain are loaded in place, or copied into a private location and loaded from there.
- **ConfigurationFile** identifies the XML configuration file that is used to configure the domain.

- **PrivateBinPath** gives a semicolon-separated list of subdirectories below the Application Base that are probed for assemblies, in addition to the Microsoft defaults.
- **PrivateBinPathProbe** determines whether the Application Base directory is probed for assemblies.



## IBM Integration Bus .NET API and documentation

[Direct Link](#)

IBM Integration Bus

- IBM.Broker.Plugin Namespace
- IBM.Broker.Plugin.Connector Namespace**
  - DataValidation Enumeration
  - NBByteArrayInputEvent Class
  - NBByteArrayInputRecord Class
  - NBByteArrayPollingResult Class
  - NBConnector Class
  - NBConnectorFactory Class
  - NBElementInputRecord Class
  - NBEVENT Class
  - NBEventInputConnector Class
  - NBInputConnector Class
  - NBInputRecord Class
  - NBPollingInputConnector Class
  - NBPollingResult Class
  - NBTimeoutPollingResult Class

**IBM.Broker.Plugin.Connector Namespace**

This namespace provides the classes for creating a .NET Connectors for .NET Input nodes.

**Classes**

Class	Description
NBByteArrayInputEvent	NBByteArrayInputEvent provides a default class to handle byte array based events.
NBByteArrayInputRecord	NBByteArrayInputRecord provides a default class to handle input records for polling results and events.
NBByteArrayPollingResult	NBByteArrayPollingResult provides a default class to handle byte array based polling results.
NBConnector	An abstract base class that represents a connector.
NBConnectorFactory	NBConnectorFactory is an abstract base class for connector factories.
NBElementInputRecord	NBElementInputRecord provides a default class to handle input records for polling results and events.
NBEvent	NBEvent is a base class which must be extended to be able to receive an event from the system.
NBEventInputConnector	NBEventInputConnector is the base class which must be extended to implement a connector which can receive events from the system.
NBInputConnector	NBInputConnector is an abstract base class which should not be extended by the user, but is extended by the system.
NBInputRecord	NBInputRecord is an abstract base class for records returned from <code>NBEvent.BuildInputRecord</code> or <code>NBPollingResult.BuildInputRecord</code> .
NBPollingInputConnector	NBPollingInputConnector is the base class which must be extended to implement a connector which can receive data from the system.
NBPollingResult	NBPollingResult is the base class for a polling result from the system.
NBTimeoutPollingResult	NBTimeoutPollingResult is a class which is used to indicate the <code>ReadData</code> method has no data to return.

© Copyright IBM Corporation 2016

Figure 2-16. IBM Integration Bus .NET API and documentation

WM676/ZM6761.0

### Notes:

The IBM Knowledge Center for IBM Integration Bus contains documentation for the .NET API, which includes:

- **IBM.Broker.Plugin Namespace.** This namespace provides the classes for creating a .NET Compute node and other helper code.
- **IBM.Broker.Plugin.Connector Namespace.** This name provides the classes for creating a .NET connector for .NET Input nodes.

## Deploying a .NET assembly

- .NET assembly contains the runtime code that the .NETCompute node calls and must be accessible to the integration node at run time
- .NET assembly can be included in the BAR file
- If the .NET assembly is not included in the BAR file, and it is loaded directly from the integration node file system, the assembly is found in one of the following ways:
  - Absolute directory path that is specified on a .NET node **Assembly name** property, or specified on an ESQL function or procedure
  - An override to the absolute directory path in a BAR file for the .NET node
  - A **DotNetAppDomain** configurable service that overrides the absolute directory path
  - Using advanced properties to search the global assembly cache

© Copyright IBM Corporation 2016

Figure 2-17. Deploying a .NET assembly

WM676/ZM6761.0

### Notes:

.NET assemblies can be included in a BAR file, or loaded directly from the integration node file system.

Include the .NET assembly in the BAR file by packaging it in a .NET application domain. The application domain is packaged as a `.appdomainzip` file at the root level of the BAR file. The .NET assembly is deployed with any other resources in the application domain.

If the .NET application domain is associated with an application, you can deploy the referencing application. The .NET assemblies in the application domain are deployed with the application. Alternatively, you can deploy the .NET application domain separately.

If the .NET assembly is not included in the BAR file, and is loaded directly from the integration node file system, the assembly is found in one of the following ways:

- The absolute directory path that is specified on a .NET node **Assembly name** property, or specified on an ESQL function or procedure
- An override to the absolute directory path in a BAR file for the .NET node
- A **DotNetAppDomain** configurable service that overrides the absolute directory path



## Unit summary

Having completed this unit, you should be able to:

- Use .NET applications on Windows integration nodes to create, route, and transform messages

© Copyright IBM Corporation 2016

---

Figure 2-18. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: A Windows .NET assembly can be packaged and deployed with the BAR file that contains the .NETCompute node message flow.

© Copyright IBM Corporation 2016

Figure 2-19. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answer here:

1.



## Checkpoint answers

1. True or false: A Windows .NET assembly is packaged and deployed with the BAR file that contains the .NETCompute node message flow.

Answer: **True**

© Copyright IBM Corporation 2016

---

Figure 2-20. Checkpoint answers

WM676/ZM6761.0

### Notes:

# Unit 3. Transforming data with XSL stylesheets

## What this unit is about

You can use the XSL Transform node to transform an XML message to another form of message, according to the rules provided by an Extensible Stylesheet Language (XSL) stylesheet. This unit describes how to use XSL stylesheets to transform data.

## What you should be able to do

After completing this unit, you should be able to:

- Transform XML data to another form of XML data, according to the rules provided by an XSL stylesheet

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Transform XML data to another form of XML data, according to the rules provided by an XSL stylesheet

© Copyright IBM Corporation 2016

---

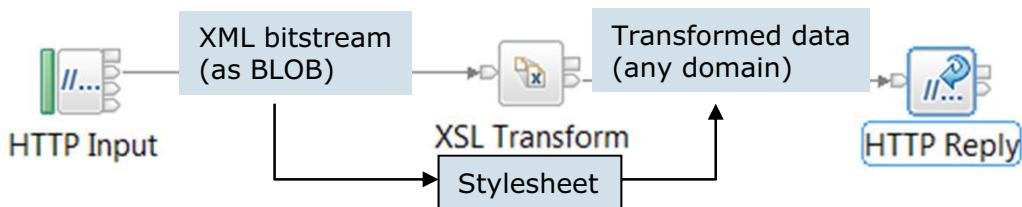
Figure 3-1. Unit objectives

WM676/ZM6761.0

### Notes:

## XSL Transform node

Transforms an XML message to another form of message, according to the rules provided by an XSL stylesheet



```

.xml  <?xml version="1.0" encoding="UTF-8"?>
      <message>Transformed using XSL!</message>
+
.xsl  <?xml version="1.0" encoding="UTF-8"?>
      <xsl:stylesheet version="1.0"
                      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
          <xsl:output method="text" encoding="UTF-8"/>
          <!-- match the contents of the message tag and copy the text out -->
          <xsl:template match="/">
              <xsl:value-of select="message"/>
          </xsl:template>
      </xsl:stylesheet>
=
      Transformed data by using XSL.
  
```

© Copyright IBM Corporation 2016

Figure 3-2. XSL Transform node

WM676/ZM6761.0

### Notes:

Extensible Stylesheet Language Transformations (XSLT) is an XML-based language that is used to transform XML documents into other XML or “human-readable” documents. The original document is not changed; rather, a new document is created based on the content of an existing one.

The IBM Integration Bus XSLTransform node transforms an XML message according to the rules provided by an XSLT stylesheet. This capability is delivered through the XALAN-4J XSLT stylesheet transformation engine. The message flow can create the output document in standard XML syntax or in another format.

The figure shows an example of an XSLTransform node in a message flow and an example of a stylesheet.

An example of a message flow with the XSLTransform node is provided in the Integration Toolkit Tutorials Gallery.

## XSL Transform node properties

- **Stylesheet name** is the name of principal stylesheet
  - Click **Browse** to select stylesheet and automatically add it to the BAR file when you add a message flow to a BAR file
  - To identify a deployed stylesheet, set the **Stylesheet name** property and leave the **Stylesheet directory** property blank
  - In the Message Flow editor, drag an `.xslt` file onto the XSLTransform node to set the stylesheet name automatically
- **Advanced** properties
  - **Stylesheet directory** is the path where the stylesheet is located
  - **Stylesheet cache level** is the number of compiled or parsed stylesheets that are stored in this instance of the node
- To ensure that the XSLTransform node transforms the message correctly, set the **Retain mixed content**, **Retain comments**, and **Retain processing instructions** properties on the preceding node

© Copyright IBM Corporation 2016

Figure 3-3. XSL Transform node properties

WM676/ZM6761.0

### Notes:

The name of the stylesheet is used when the stylesheet specification is searched for in node properties. To specify a stylesheet by using node properties, enter the required value for **Stylesheet name**.

Specify a principal stylesheet by using one of the following methods:

- Click **Browse** next to **Stylesheet name** and then select the stylesheet from the file system. The identified principal stylesheet and all its relatively referenced descendant stylesheets are added automatically to the BAR file when you add a message flow to a BAR file. The descendant stylesheets are added only if both they and their parent stylesheets are available.
- To identify an already deployed, or ready to be deployed, stylesheet, use the **Stylesheet name** property, and leave the **Stylesheet directory** property blank.
- In the Message Flow editor, drag an `.xslt` file onto the XSL Transform node to set the **Stylesheet name** automatically.

An XSLT compiler is used for the transformation if the stylesheet is not embedded within the message, and the node cache level (node property **Stylesheet Cache Level**) is greater than zero.

If the XSLT is cached, the performance is improved because the XSLT is not parsed every time that it is used.

If the input to the XSL Transform node is generated from the XMLNSC parser, the XMLNSC parser discards certain information in XML documents, such as processing instructions and comments. However, you can set properties to retain this information in a preceding node. To ensure that the XSLTransform node transforms the message correctly, set the **Retain mixed content**, **Retain comments**, and **Retain processing instructions** properties correctly on the preceding node (for example, an MQInput node).

The MRM parser also discards the XML processing instructions and comments. If this information is important for your transformation, refrain from using the MRM domain.

## Deployed and non-deployed stylesheets

- Deployed stylesheets
  - Imported into a BAR file and deployed to target systems
  - Managed by the integration node
- Non-deployed stylesheets
  - Stored in a location where the XSL Transform node can access them
  - Not managed by the integration node

© Copyright IBM Corporation 2016

Figure 3-4. Deployed and non-deployed stylesheets

WM676/ZM6761.0

### Notes:

You can use stylesheets in two different ways with the XSLTransform node.

*Deployed stylesheets* are stylesheets that you import into a BAR file and deploy to target systems. The integration node manages deployed stylesheets.

*Non-deployed stylesheets* are stylesheets that you store in a location where the XSLTransform node can access them. The integration node does not manage non-deployed stylesheets.

## Before you deploy stylesheets

- Stylesheets must have either .xsl or .xslt file extension
- Import stylesheets into the Eclipse workspace
  - Put location-dependent descendant stylesheets in the correct directory structure relative to their parent stylesheets
  - Do not put location-dependent descendants in the Eclipse workspace that you do not want to deploy
- If the **Stylesheet name** property on an XSL Transform node is not specified, ensure that references to the stylesheet files are relative

© Copyright IBM Corporation 2016

Figure 3-5. Before you deploy stylesheets

WM676/ZM6761.0

### Notes:

To use deployed stylesheets, you must complete the following prerequisite steps.

1. Make sure that the files have the correct file name extensions. Deployed stylesheets must have either .xsl or .xslt as their file extension.
2. Import into an Eclipse workspace project all stylesheets that are going to be deployed. Put location-dependent descendant stylesheets in the correct directory structure relative to their parent stylesheets. Do not put in the Eclipse workspace location-dependent descendants that you do not want to deploy.
3. Typically, all references to a deployed stylesheet must be made relative, no matter where they are displayed. A reference to a principal stylesheet must be made relative to the root of the relevant Eclipse workspace project.

The only exception is when you specify a principal stylesheet as the **Stylesheet name** property on an XSL Transform node. You can use an absolute path that points to the correct directory structure in the Eclipse workspace. If the principal stylesheet is found, the system resets the node property automatically to the correct relative value.

## Stylesheet search order

XSL Transform node searches for the XSL file in the following order:

1. Dynamically from the input message

Example: `<?xml-stylesheet type="text/xsl" href="abc.xsl"?>`

2. Dynamically from the `XSL.StyleSheetName` local environment variable

- Allows for dynamic selection of the stylesheet from within the message flow
- Can apply multiple stylesheets to one message

3. Statically from **Stylesheet name** and **Stylesheet directory** in the XSL Transform node properties

© Copyright IBM Corporation 2016

Figure 3-6. Stylesheet search order

WM676/ZM6761.0

### Notes:

The XSL Transform node supports different ways to specify the name of the stylesheet file and its location:

- As part of the XML data that is contained in the message, which allows the node to apply different stylesheets to the XML data it receives
- In the **LocalEnvironment** folder in the logical message tree, which allows the node to apply different stylesheets to the XML data it receives
- In the node properties, which assure that the node always applies the same stylesheet to all the XML data it receives

The node then serializes the message body, gets the XML data from the message, and applies the stylesheet to the XML data to produce new data. Then, the newly created data is placed in the message, and the message is sent to the next node.

## Working with XML messages and the logical message tree



Figure 3-7. Working with XML messages and the logical message tree

WM676/ZM6761.0

### Notes:

To work with XML messages, it is necessary to understand some common syntax element types. This figure shows some of the common syntax elements and the logical message tree.

**Element (tag)** is the default name element. It can have many subcomponents, including attributes, elements, and content.

**Attribute (attr)** is the default name-value element. Attributes must have a parent element, but must not have subelements.

**Content (pcdata)** is the default value element that represents character data (including white space).

In an Extended Structured Query Language (ESQL) program, you can refer to field types by their symbolic names or by their hex or numeric expression. In Java, the `MbElement` class provides methods to get and set the parser-specific type.

All fields are considered character for the Integration Bus XML parsers. If you intend to use a field in a compute expression, change the data type associated with it.

## Determining the output message format

- XSL Transform node searches for the message domain, message model, message type, and message format to use for the output message by interrogating, in the following order:
  1. Dynamically from `XSL.MessageDomain`, `XSL.MessageSet`, `XSL.MessageType`, and `XSL.MessageFormat` local environment variables
  2. XSL Transform **Output Message Parsing** node properties
- BLOB domain is used if the node cannot determine the message domain from the `XSL.MessageDomain` environment variable or the **Message domain** property

© Copyright IBM Corporation 2016

Figure 3-8. Determining the output message format

WM676/ZM6761.0

### Notes:

The LocalEnvironment variables in the incoming message and the **Output Message Parsing** properties that are specified in the node determine the output message format from the XSL Transform node.

If no output domain can be determined, the BLOB domain is used.

## Unit summary

Having completed this unit, you should be able to:

- Transform XML data to another form of XML data, according to the rules provided by an XSL stylesheet

© Copyright IBM Corporation 2016

Figure 3-9. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or false: An XSL Transform can be used for an input message in the comma-separated values (CSV) format.
2. True or false: The stylesheet that is referenced in the XSL Transform node must be deployed with the message flow in the BAR file.

© Copyright IBM Corporation 2016

Figure 3-10. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.

## Checkpoint answers

- True or false: An XSLTransform can be used for an input message in CSV (comma-separated values) format.

Answer: **False**. The input format for any XML stylesheet must be a well-formed XML document.

- True or false: The stylesheet that is referenced in the XSL Transform node must be deployed with the message flow in the BAR file.

Answer: **False**. It is not necessary to deploy the stylesheet. The stylesheet can be stored in a location where the XSL Transform node can access them.

© Copyright IBM Corporation 2016

Figure 3-11. Checkpoint answers

WM676/ZM6761.0

### Notes:



# Unit 4. Analyzing XML documents

## What this unit is about

The IBM Integration Bus Data Analysis function analyzes and filters information in complex XML documents. You can use the analysis to create a library that contains Data Analysis tools to quickly and easily transform the data in IBM Integration Bus. This unit describes the Data Analysis tools that are available in the IBM Integration Toolkit for analyzing XML data.

## What you should be able to do

After completing this unit, you should be able to:

- Use the IBM Integration Toolkit Data Analysis perspective to analyze and filter information in complex XML documents

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center for IBM Integration Bus



## Unit objectives

After completing this unit, you should be able to:

- Use the IBM Integration Toolkit Data Analysis perspective to analyze and filter information in complex XML documents

© Copyright IBM Corporation 2016

---

Figure 4-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Using XML data

The problem:

- XML documents are getting bigger
- XML documents are getting more complex
- Non-descriptive XML schema definitions are poorly constructed either accidentally or deliberately in an attempt to construct more dynamic integration capabilities

The solution:

- The Integration Toolkit Data Analysis perspective provides a set of models, views, and tools that analyze and filter information in complex XML documents
- Use data analysis to model your data, and create libraries, maps, and subflows to easily transform it within the Integration Bus

© Copyright IBM Corporation 2016

Figure 4-2. Using XML data

WM676/ZM6761.0

### Notes:

XML files are flexible, but can be long and complex. You might be interested in a small subset of data. However, it can be difficult to construct a model to filter and analyze this subset of data by traditional methods. Existing business intelligence tools work best with well-structured relational data.

The Data Analysis perspective in the Integration Toolkit analyzes and filters information in complex XML documents. You can use this analysis to create a library that contains Data Analysis tools to quickly and easily transform your data in Integration Bus.



## Data analysis overview

- **Data Analysis model** is populated based on the analysis of groups of XML documents and iteratively improved as more documents are added
- **Target model** can be constructed by dragging and dropping from the Data Analysis model
- Graphical maps and subflows are generated from the target model
- Optionally, you can generate a map for inserting data into a relational database

© Copyright IBM Corporation 2016

Figure 4-3. Data analysis overview

WM676/ZM6761.0

### Notes:

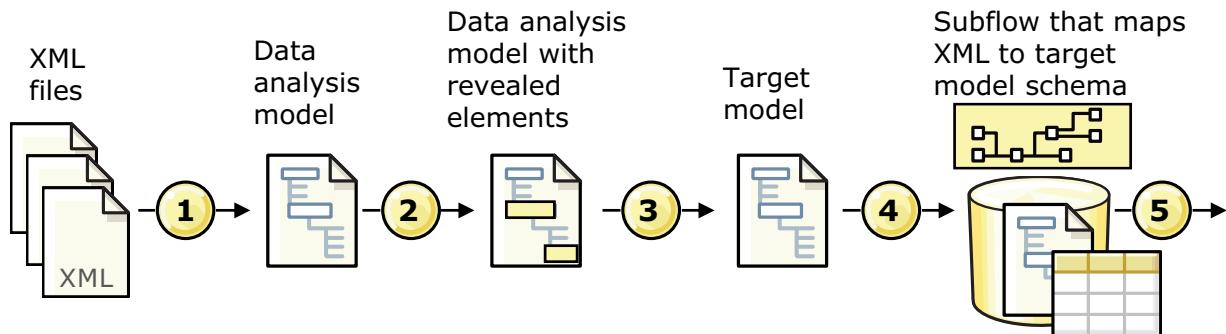
In a Data Analysis project, you analyze a set of sample XML documents according to the content of the data. This content is defined in a Data Analysis profile that you specify when you create the project.

When you analyze your sample XML documents, new data and new data structures within them are retained in the *Data Analysis model*. Repeated data and data structures are not added.

After the analysis is complete, you can select relevant in the Data Analysis model, and then add them to a *target model*.

You can use the target model to produce a library that contains Data Analysis tools that you can use at run time to transform incoming data. Data Analysis tools include a map, schema file, validation stylesheet, and subflow.

## Data analysis process



1. Integration Bus generates a Data Analysis model from XML files.
2. Views and filters help you move through the complex content.
3. Revealed elements, whose content relates to a known code set translation, are highlighted and used to generate a target model.
4. Make further edits to the target model (either for output messages or output to a database) and generate graphical maps that convert input instance XML documents into instance XML documents.
5. Use the generated subflow and associated resources in the Integration Toolkit.

© Copyright IBM Corporation 2016

Figure 4-4. Data analysis process

WM676/ZM6761.0

### Notes:

The key steps for analyzing XML documents are:

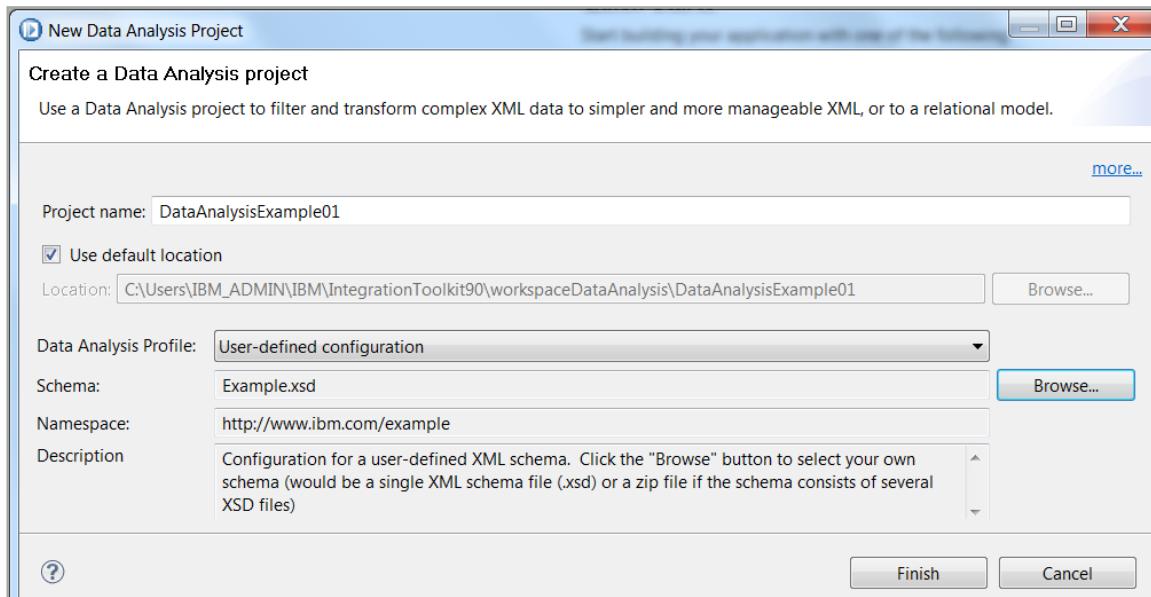
1. Create a Data Analysis project.
2. Analyze sample XML documents.
3. Create a target model.
4. Populate and edit your target model.
5. View and edit your target model in database format.

You can also create a library that contains Data Analysis tools and the Data Analysis tools to work with your data in the Integration Toolkit Integration Development perspective.

The Data Analysis tools are all grouped into a single Data Analysis perspective, which consists of a set of coordinated views. Selecting an element in one view moves you to the relevant part of another view.

# WebSphere Education

## Creating a Data Analysis project



- Data Analysis projects must use a Data Analysis profile
  - User-defined configuration Data Analysis profile
  - IBM Integration Bus Healthcare Pack Data Analysis profiles
  - Book Series (sample configuration)

© Copyright IBM Corporation 2016

Figure 4-5. Creating a Data Analysis project

WM676/ZM6761.0

### Notes:

The Data Analysis perspective is driven from a Data Analysis project. A Data Analysis project is used to create, contain, and develop your Data Analysis model, target models, and Data Analysis tools.

To create a Data Analysis project:

1. In the IBM Integration Toolkit, click **File > New > Data Analysis Project**. The **New Data Analysis Project** wizard opens.
2. Complete the fields in the **New Data Analysis Project** wizard and click **Finish**. The Data Analysis perspective and the Data Analysis Project Overview open.

In a Data Analysis project, you analyze a set of sample XML documents according to the content of the data. This content is defined in a *Data Analysis profile* that you specify when you create the project. The Data Analysis profile contains the schema that provides a common definition against which all the instance documents successfully validate. You can create your own profiles or use predefined profiles from the IBM Integration Bus Healthcare Pack to analyze XML data.

## IBM Integration Bus Healthcare Pack Data Analysis profiles

- IBM Integration Bus Healthcare Pack provides Data Analysis profiles to analyze healthcare data that is routed through message flows
  - Use the HL7 v2 Data Analysis profile to analyze Health Level 7 (HL7) data
  - Use the HL7 CDA Data Analysis profile to analyze HL7 Clinical Document Architecture (CDA) documents
  - Use the HL7 v2 (ORU) Data Analysis profile to analyze HL7 observation result (ORU) messages
  - Use the DICOM Data Analysis profile to analyze Digital Imaging and Communications in Medicine (DICOM) XML data

© Copyright IBM Corporation 2016

Figure 4-6. IBM Integration Bus Healthcare Pack Data Analysis profiles

WM676/ZM6761.0

### Notes:

IBM Integration Bus Healthcare Pack builds on IBM Integration Bus to support applications in healthcare.

You can use an Integration Bus Healthcare Pack Data Analysis profile to analyze and filter healthcare data that is contained in XML files. This figure lists the profiles that the Integration Bus Healthcare Pack provides.

**Data Analysis Project Overview**

Use a Data Analysis project to filter and transform complex XML data to simpler and more manageable XML, or to a relational model. [more...](#)

**General**

Project Name: [DataAnalysisExample01](#)  
Schema File: [Example.xsd](#)

**Tasks**

- Analyze representative sample documents. The toolkit analyzes and transforms them into a common data model. You then use the common data model to specify your target model.  
[Analyze documents...](#)
- After you analyze your documents, specify a target model. Create a model, or load an existing one from the table below.  
[Create a new target model...](#)
- After you create a valid target model, you can use the Target Model editor to your target model, and place them in an application or library.

Model	Description

[Open model](#)  
[Delete model](#)

**Overview**

Data Analysis Project Overview editor opens when you open the Data Analysis project file (.dap) that is associated with the Data Analysis project

© Copyright IBM Corporation 2016

Figure 4-7. Data Analysis Project Overview editor

WM676/ZM6761.0

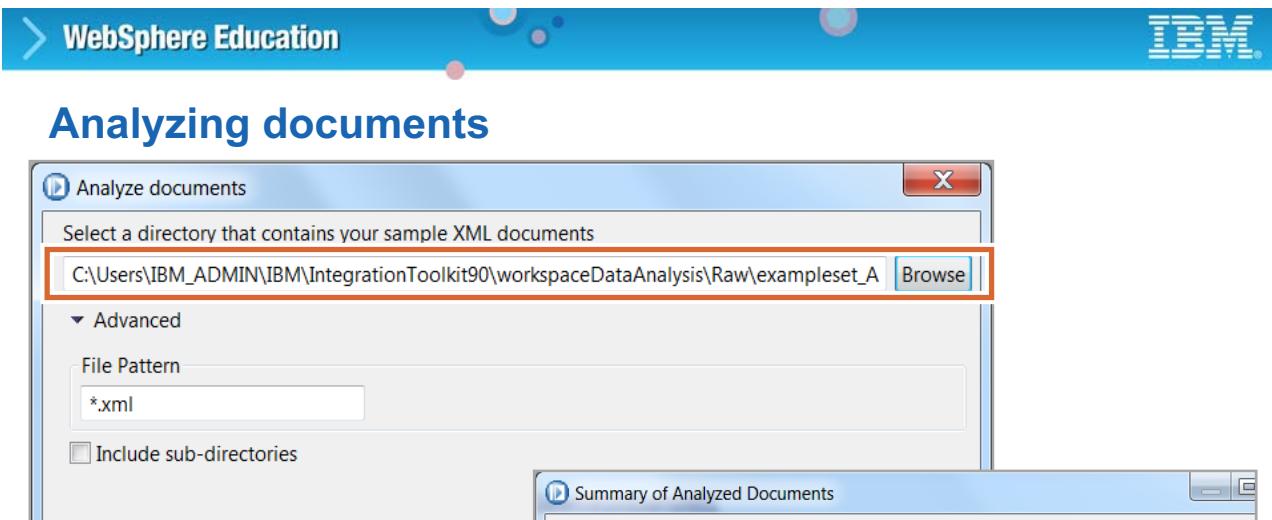
## Notes:

By default, the Editor view in the Data Analysis perspective contains the Data Analysis Project Overview and Target Model editors.

You use the Data Analysis Project Overview editor to analyze sample XML documents, and to add data structures that conform to your schema into a Data Analysis model.

You select elements and attributes from your Data Analysis model to form a target model, which you use to create a library that contains Data Analysis tools.

To start analyzing XML documents, click **Analyze documents** in the Data Analysis Project Overview editor.



- After project creation, analyze one or more XML instance documents, by specifying the directory where the Data Analysis Project Overview opens when the Data Analysis project file is opened

© Copyright IBM Corporation 2016

Figure 4-8. Analyzing documents

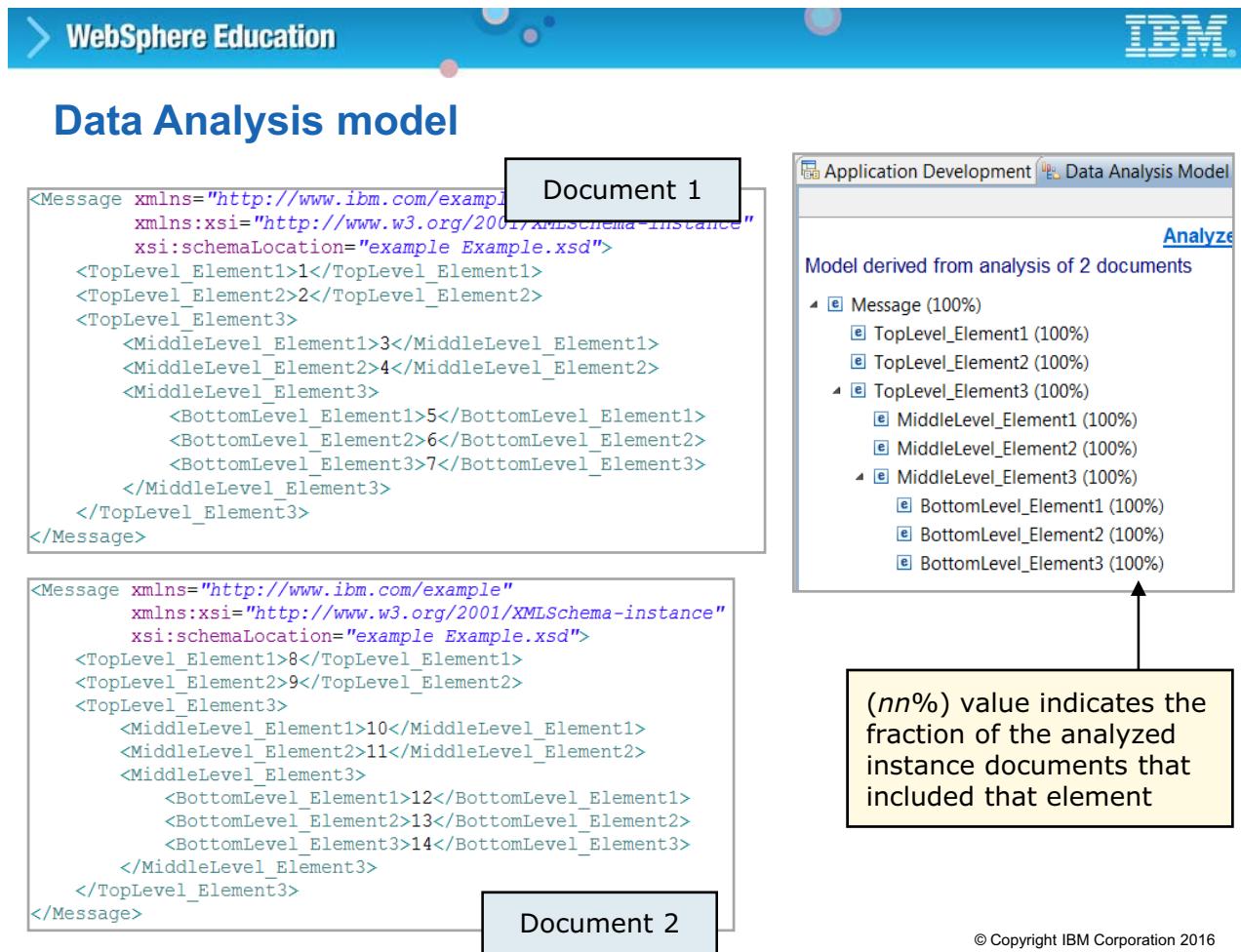
WM676/ZM6761.0

### Notes:

The next step for analyzing XML documents is to select a directory that contains the sample XML documents.

First, ensure that your sample documents are in the XML format. The sample XML documents must be representative of your relevant data so that you can later produce a useful target model. If data is missing from your sample XML documents, subsequent input might not conform to the generated target model.

Second, ensure that the sample XML documents conform to a schema and that the schema is available. If the sample data does not conform to your schema, you cannot analyze your sample XML documents, nor load the relevant data from within them.



© Copyright IBM Corporation 2016

Figure 4-9. Data Analysis model

WM676/ZM6761.0

## Notes:

The Data Analysis example in the figure uses a single XML schema to which two separate XML instance documents conform.

The **Document instance percentage** icon can be selected to show the percentages in brackets at each level in the hierarchy of the Data Analysis model. The percentages identify the percentage of the instance documents that were analyzed that include the element in question.



## Data Filter view

Use the **Data Filter** view to search the Data Analysis model

Select a row in the **Data Filter** view to shows its **Properties** and **Data Path**

Element Name	Element Data	Element Type	Depth of Descendant [min, max]	Docun
TopLevel_Element1	TopLevel_Element1	e string	[0,0]	3
MiddleLevel_Element1	MiddleLevel_Element1	e string	[0,0]	2
BottomLevel_Element1	BottomLevel_Element1	e string	[0,0]	2

Properties View:

Property	Value
General	
Data Path	MiddleLevel_Element1
Element Data	MiddleLevel_Element1
Element Name	MiddleLevel_Element1
Element Namespace URI	http://www.ibm.com/example
Statistics	
Document count	2
Total count	2
Type	
XML Schema Type	string
XML Schema Type Namespace	http://www.w3.org/2001/XMLSchema

Data Paths View:

MiddleLevel\_Element1 (Message/TopLevel\_Element3/)

© Copyright IBM Corporation 2016

Figure 4-10. Data Filter view

WM676/ZM6761.0

### Notes:

The **Data Filter** view shows the elements that are added from the sample XML documents. The elements are organized according to static cues in the structure and are listed alphabetically by **Element Name**.

You can filter the results and sort the order according to **Element Name** or **Element Data**.

When you click an element in the **Data Filter** view, the **Properties** view and the **Data Paths** view update to show details of the selected element.



## Depth of descendants

The screenshot shows the Data Filter view in WebSphere Studio. The left pane displays a tree of XML nodes, and the right pane shows a table of element occurrences with columns for Element Name, Element Data, Element Type, and Depth of Descendant [min, max]. A yellow box labeled "Level 0" highlights the first occurrence of the Volume element, which has a depth of 1. Another yellow box labeled "Level 4" highlights the "entry" element under the Name element, which has a depth of 4.

Element Name	Element Data	Element Type	Depth of Descendant [min, max]
volumeInfo	volumeInfo	VolumeInfo	[1,1]
Volume	Volume	Volume	[4,4]
VolPubLoc	VolPubLoc	string	[0,0]

- **Depth of Descendant** column specifies the number of levels that are contained in all occurrences of the element

© Copyright IBM Corporation 2016

Figure 4-11. Depth of descendants

WM676/ZM6761.0

### Notes:

The **Depth of Descendant** column in the **Data Filter** view specifies the number of levels of descendant it contains for each of its occurrences in the model.

In this example, all examples of the element that are named **Volume** have four levels of descendants (4, 4). So, the minimum and maximum values of the **Depth of Descendant** column are the same. If some occurrences of the **Volume** element had fewer descendants, you would see different values for the minimum and maximum.



## Highlight all coexisting elements

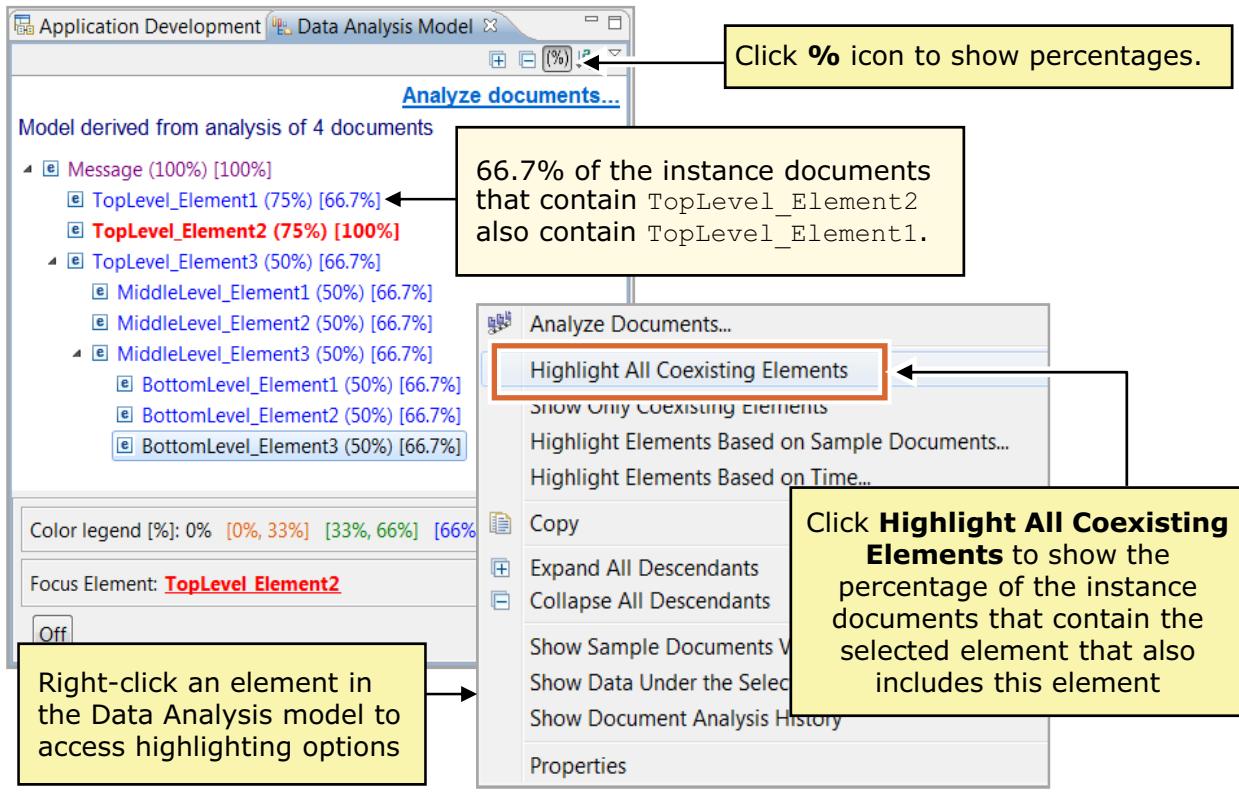


Figure 4-12. Highlight all coexisting elements

WM676/ZM6761.0

### Notes:

In this example, a fourth instance of an XML document was added to the analysis.

Clicking **Highlight All Coexisting Elements** highlights all elements that are in one or more of the same sample XML documents as your selected element. This action shows where combinations of data structures occur.

The selected element is highlighted in the **Data Analysis Model** view, and is also identified at the bottom of the view, as the **Focus Element**. The percentage next to each element indicates the percentage of documents that contain both that element and the focus element.

The screenshot shows the 'Preferences' window of the WebSphere Education interface. On the left, there's a sidebar with a 'type filter text' input field and a list of categories: Integration Development, Build Broker Archive, Data Analysis, Diagnostic Logging, Glossary File Lookup Services, Delimited Glossary File Lookup Service, and XML Glossary File Lookup Service. The 'XML Glossary File Lookup Service' option is highlighted with a red box. On the right, a main panel titled 'XML Glossary File Lookup Service' shows a table with one row: File Location/Contributor Plugin ID (com.ibm.hcls.sdg.bookseries), Glossary Reference (BookCodes), and Description (Book Series Terminology File). Below this is a 'View Content' window displaying the following XML code:

```

<?xml version="1.0" encoding="UTF-8"?>
<codeSet>
  <code>
    <codeSystem>BookCodes</codeSystem>
    <entry>
      <keyValue name='secType'>MONO</keyValue>
      <displayName>Monograph</displayName>
    </entry>
    <entry>
      <keyValue name='secType'>SURV</keyValue>
      <displayName>Survey</displayName>
    </entry>
    <entry>
      <keyValue name='mediaType'>DVD</keyValue>
      <displayName>Optical Disc</displayName>
    </entry>
    <entry>
      <keyValue name='mediaType'>SW</keyValue>
      <displayName>Software</displayName>
    </entry>
  </code>
</codeSet>

```

© Copyright IBM Corporation 2016

Figure 4-13. XML Glossary File Lookup Service

WM676/ZM6761.0

## Notes:

Data element code in your sample XML documents can be difficult to understand because it often consists of strings of numbers rather than descriptive words. Glossary files contain codes for each data element and their equivalent names. The Glossary File Lookup Service retrieves the names, and shows them instead of the data element codes in your Data Analysis project.

The Glossary File Lookup Service makes it easier to use Data Analysis because it is simpler to understand the names than their equivalent data element codes.

The Glossary File Lookup Service is accessed by selecting **Window > Preferences > Integration Development > Data Analysis > Glossary File Lookup Service**.



## Refining a target model

When you finish editing the target model, you can generate the mapping and workflow artifacts to be deployed to the Broker. [Generate...](#)

Element	Type
Editor	[1..1]
Volume	[0..*]
Title	[0..1]
Video Disc	[1..1]
media type	
Title	[1..1]
Volume Date	[1..1]
AdditionalTrailer	
SummaryStatement	

Element symbols with the shortcut arrow in the lower left corner originate directly from the Data Analysis model

You can add your own elements to the target model

© Copyright IBM Corporation 2016

Figure 4-14. Refining a target model

WM676/ZM6761.0

### Notes:

You can use the Target Model editor to alter the structure of your target model to meet the requirements of the downstream application.

With the Target Model editor you can:

- Create local elements
- Rename elements and attributes
- Move elements
- Delete elements and attributes
- Restore deleted elements and attributes

WebSphere Education

## Creating Data Analysis tools

Generate Data Analysis Tools

**Destination of your tools**

Create a Message Broker library to store your Data Analysis tools and, if required, validate your input documents

Library

Validation of input messages

Your library contains a subflow.

Your subflow contains a map to transform your input documents to your target model, an output schema describing your target model and optionally, a validation stylesheet to validate your input documents.

- Use the target model to create a library, which contains the following Data Analysis tools:
  - Schemas
  - Maps
  - Subflows
  - Validation stylesheets
- In the Target Model editor, click **Generate** to open the Generate Data Analysis Tools wizard

© Copyright IBM Corporation 2016

Figure 4-15. Creating Data Analysis tools

WM676/ZM6761.0

### Notes:

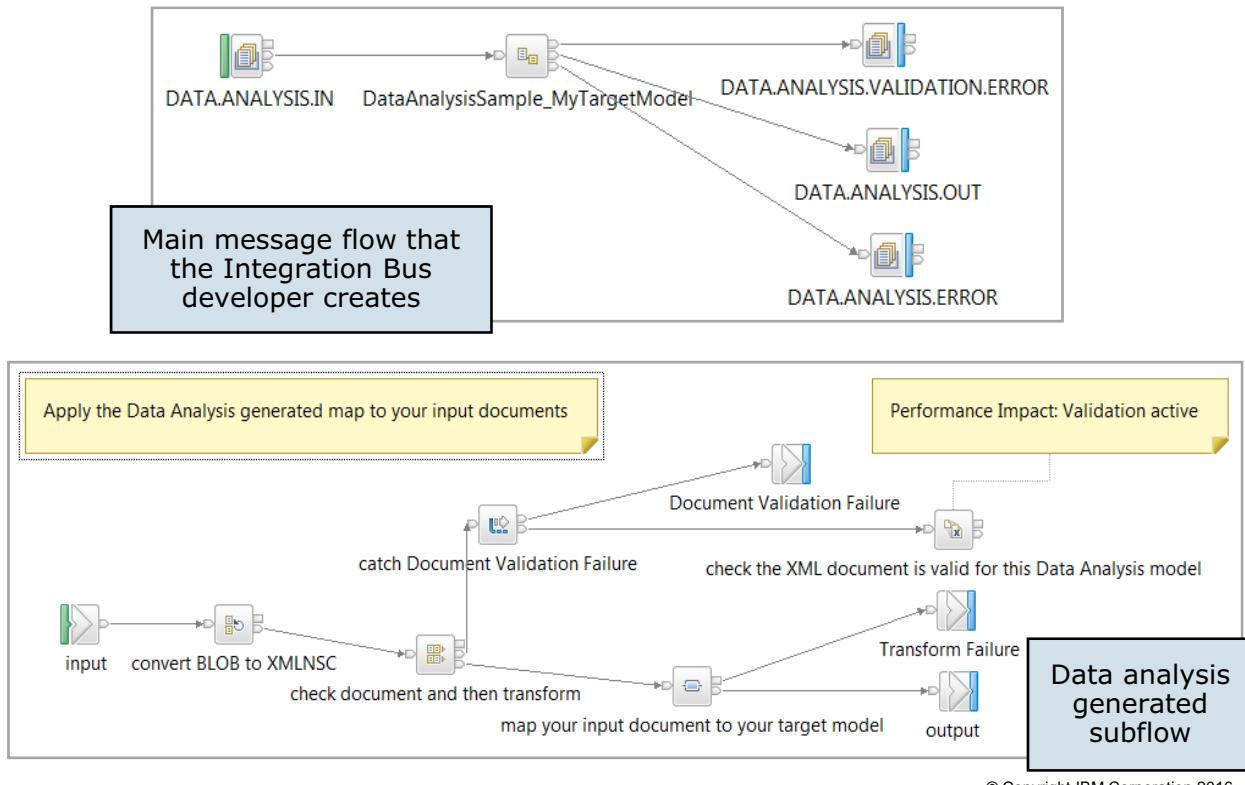
You can use your target model to create a library, which contains the following Data Analysis tools:

- XML schemas
- Integration Bus maps
- Integration Bus subflows
- Validation stylesheets

To generate the Data Analysis tools, click **Generate** in the Target Model editor.

A library that contains your Data Analysis tools is created. The subflow opens in the Message Flow editor.

## Generated subflow



© Copyright IBM Corporation 2016

Figure 4-16. Generated subflow

WM676/ZM6761.0

### Notes:

The Data Analysis subflow can be used to validate and transform a message against the target model that you create.

The subflow first converts the incoming message to XML. If the message is valid, it then maps the message to your target model. The subflow also includes some error handling logic for identifying validation failures and writing them to a queue.

Generate...' A table below shows the target model structure with columns for Table, Column, and SQL Data Type."/>

	Table	Column	SQL Data Type
MyTargetModel	[1..1]	MYTARGETMODEL	---
Editors	[0..1]	EDITORS	---
Editor	[1..*]	EDITOR	VARCHAR(256)
Volume	[1..*]	VOLUME	
Title	[1..1]	TITLE	VARCHAR(256)
Video Disc	[0..*]	VIDEO_DISC	
mediaType	[0..1]	MEDIATYPE	VARCHAR(256)
Title	[1..1]	TITLE	VARCHAR(256)
VolPubDate	[1..1]	TITLE_VOLPUBDATE	VARCHAR(256)
AdditionalTrailer	[0..1]	ADDITIONALTRAILER	---
SummaryStatement	[0..1]	SUMMARYSTATEMENT	---

To view the target model in a database format:

1. Click the **Enable relational database mapping** icon.
2. Complete the **Destination Database** and **Destination Schema** fields, and then select the product and version of your database.
3. To generate the default SQL table names for the target database, click the **Relational mapping options** icon and then click **Generate Default SQL Table Naming**.

© Copyright IBM Corporation 2016

Figure 4-17. Target model: Relational database map

WM676/ZM6761.0

## Notes:

The figure shows the steps for viewing the target model in a database format.

1. Click the **Enable relational database mapping** icon. The **Destination Database Platform Selection** wizard opens.
2. Complete the **Destination Database** and **Destination Schema** fields. Select the product and version of your database. Click **OK**. The blank **Table**, **Column**, and **SQL Data Type** columns are shown.
3. To generate the default SQL table names for your target database, click the **Relational mapping options** icon and then click **Generate Default SQL Table Naming**.

The Target Model editor is populated with the details of your target model, in a database format.

In the populated model, you can:

- Rename tables and columns
- Assign different element types to elements
- Delete cell contents

## Generated subflow that includes Database nodes

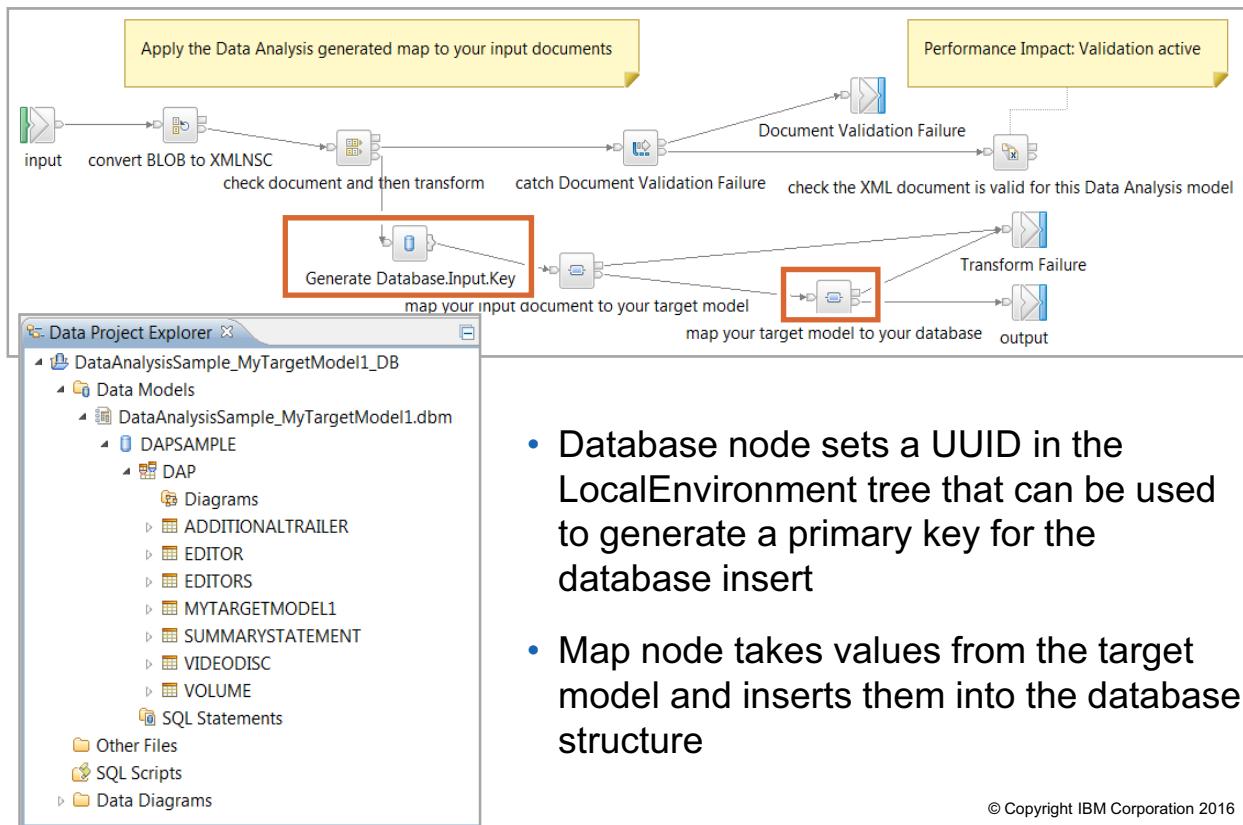


Figure 4-18. Generated subflow that includes Database nodes

WM676/ZM6761.0

### Notes:

You can use Integration Bus to insert the target model information into an existing database. Integration Bus currently supports DB2, Oracle, and Microsoft SQL Server for a Data Analysis database. The IBM Knowledge Center for Integration Bus contains instructions for setting up a Data Analysis database.

The Data Analysis perspective generates a subflow that maps the target model to the database. The message flow contains a map node that sets a UUID in the LocalEnvironment tree. The UUID can be used to generate a primary key for the database. Another map node inserts the target model into the database structure.



## Unit summary

Having completed this unit, you should be able to:

- Use the IBM Integration Toolkit Data Analysis perspective to analyze and filter information in complex XML documents

© Copyright IBM Corporation 2016

---

Figure 4-19. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: Data analysis identifies the effect of changes to IBM Integration Bus artifacts.
2. Complete the sentence: A \_\_\_\_\_ can be constructed by dragging and dropping from the Data Analysis model in the Integration Toolkit.

© Copyright IBM Corporation 2016

Figure 4-20. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.

## Checkpoint answers

1. True or false: Data analysis identifies the effect of changes to IBM Integration Bus artifacts.

Answer: **False**. Impact analysis identifies the effect of changes to IBM Integration Bus artifacts.

2. Complete the sentence: A \_\_\_\_\_ can be constructed by dragging and dropping from the Data Analysis model in the Integration Toolkit.

Answer: **Target model**

© Copyright IBM Corporation 2016

Figure 4-21. Checkpoint answers

WM676/ZM6761.0

### Notes:

# Unit 5. Modeling complex data with DFDL

## What this unit is about

You can model a wide variety of message formats by using DFDL schema files. In this unit, you learn how to use DFDL to model complex data that includes multiple records types, length prefixes, choice groups, optional elements, and variable array elements.

## What you should be able to do

After completing this unit, you should be able to:

- Reuse a DFDL model to create a more complex model
- Define length prefixes in a DFDL model
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message

## How you will check your progress

- Checkpoint
- Lab exercises

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Reuse a DFDL model to create a more complex model
- Define length prefixes in a DFDL model
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message

© Copyright IBM Corporation 2016

---

Figure 5-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- DFDL model review
- Modeling complex data
- Industry-standard models

© Copyright IBM Corporation 2016

Figure 5-2. Topics

WM676/ZM6761.0

## Notes:



## 5.1. DFDL model review

This topic is a brief review of the DFDL concepts that are described in detail in the prerequisite course, WM666: *IBM Integration Bus V10 Application Development I*.

## DFDL data support (1 of 2)

- Language structures such as COBOL, C, and PL/I
- Industry standards such as SWIFT, HL7, FIX, HIPAA, X12, EDIFACT, ISO8583
- Fixed data and text of binary markup delimited data
- Text data types such as strings, numbers, zoned decimals, calendars, Booleans
- Binary data types such as integers, floats, BCD, packed decimal, calendars, Booleans
- Bidirectional text
- Bit data of arbitrary length
- Pattern languages for text numbers and calendars
- Ordered, unordered, and floating content
- Default values on parsing and serializing

© Copyright IBM Corporation 2016

Figure 5-3. DFDL data support (1 of 2)

WM676/ZM6761.0

### Notes:

By using DFDL, you can model the types of data that IBM Integration Bus supports. The complete list of supported data types is shown on this figure and the next.

DFDL can model a superset of the data formats that are available in the message modeling components of Integration Bus. However, the traditional modeling components of IBM Integration Bus are still available. Some types of data are not amenable to being modeled with DFDL (such as XML). For these data types, you use the traditional modeling process and parsers to process the data at run time.

## DFDL data support (2 of 2)

- Nil values for handling out-of-band data
- XPath 2.0 expression language, including variables to model dynamic data
- Speculative parsing to resolve choices and optional content
- Fixed and variable arrays
- Hide elements in the data
- Calculate element values
- Validation to XML Schema 1.0 rules
- Scoping mechanism to allow common property values to be applied at multiple points

© Copyright IBM Corporation 2016

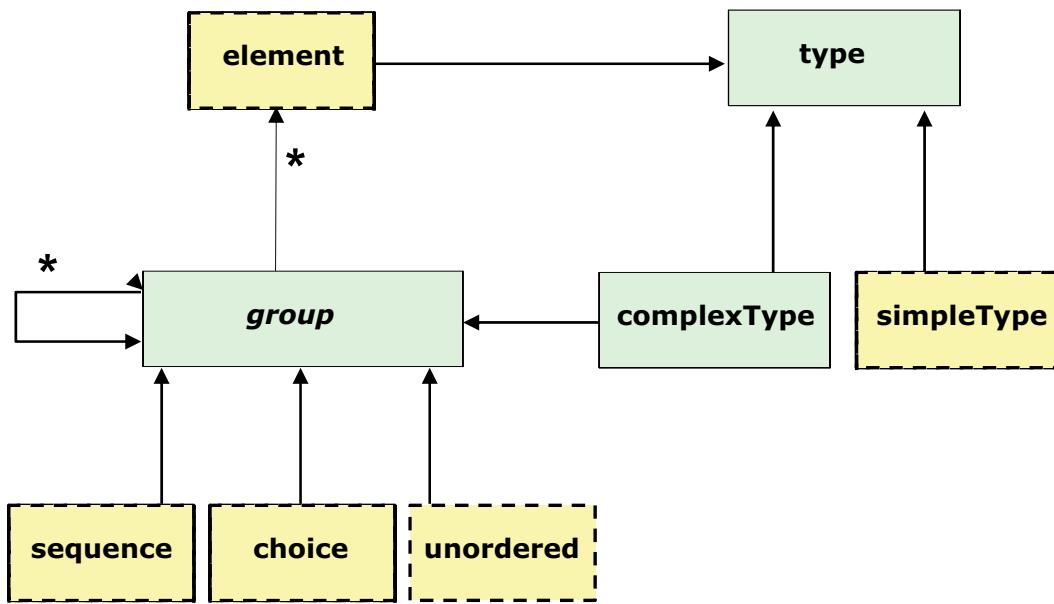
Figure 5-4. DFDL data support (2 of 2)

WM676/ZM6761.0

### Notes:

The DFDL standard provides an extensive list of supported data types. Not all of them are listed in this figure and the previous figure.

## DFDL object model



- DFDL properties are placed only on element, simpleType, sequence, and choice objects
- The asterisk (\*) indicates a zero-to-many relationship

© Copyright IBM Corporation 2016

Figure 5-5. DFDL object model

WM676/ZM6761.0

### Notes:

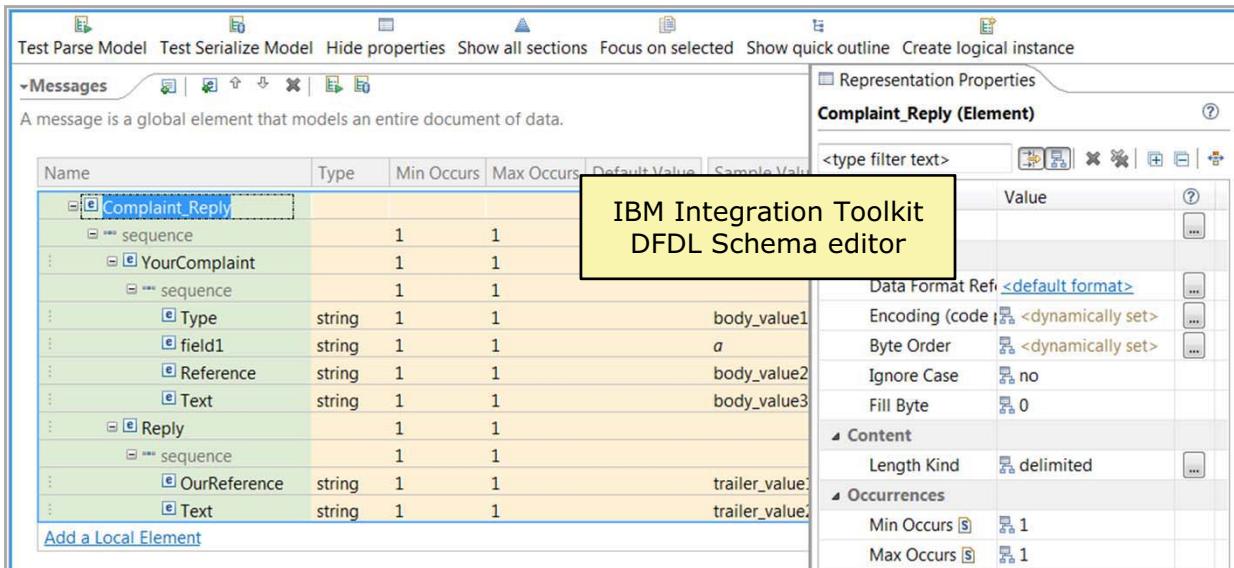
When you model with DFDL, an object model defines the logical format of the data. This figure shows the components of the DFDL object model and their relationships.

The basic component is an element. The element type defines the representation of the physical data. For example, a simple data type, such as a field, can be defined as a string or an integer. Complex elements, such as records or database rows, are also referred as groups. A sequence group has a fixed number of elements that are always in the same order in the data.

DFDL is an extension to traditional XML-based data models. An XML schema is written for the logical model of the data. The schema is augmented with special DFDL annotations, which describe the local representation of the data. These DFDL annotations (properties) are valid only on sequence, choice, element, and simple types.

## DFDL properties

- Describe the physical representation of the objects in a DFDL schema
- Do not have built-in defaults



© Copyright IBM Corporation 2016

Figure 5-6. DFDL properties

WM676/ZM6761.0

### Notes:

DFDL properties describe the physical representation of the data objects in a DFDL schema. The object determines the available properties.

- For elements and simple types, you can define the data representation and how the length of the element is determined (**Length Kind** property).
- For a sequence group, you can define the sequence type (**Sequence Kind**), such as ordered or unordered. You can also define the separator character (**Separator**) between the elements of the group.
- For all elements, you can define an initiator character or characters (**Initiator**), a terminator character or characters (**Terminator**), encoding, and whether the data is aligned to the left or to the right (**Alignment**).

It is important to remember that DFDL properties do not have built-in defaults. You should specify default values in the model.

## DFDL expressions

- Can be used to set some properties dynamically at processing time
  - When a property value must be set dynamically from the contents of the data
  - In an assert or discriminator annotation
  - When setting the value or default value of a variable
- Expression language is a subset of XPath 2.0, including variables, and with some extra DFDL-specific functions

**XPath Expression Builder**

Select the target from the Schema viewer, Function viewer or Operator viewer and drag and drop the nodes in the source viewer below.

Data Types Viewer	XPath Functions	Operators
<input checked="" type="checkbox"/> Company <input type="checkbox"/> \$fdfd:binaryFloatRep : string <input type="checkbox"/> \$fdfd:byteOrder : string <input type="checkbox"/> \$fdfd:encoding : string <input type="checkbox"/> \$fdfd:outputNewLine : string	$\text{fn:minutes-from-time}(\text{time})$ : integer $\text{fn:month-from-date}(\text{date})$ : integer $\text{fn:month-from-datetime}(\text{dateTime})$ : integer $\text{fn:seconds-from-datetime}(\text{dateTime})$ : decimal $\text{fn:seconds-from-time}(\text{time})$ : integer $\text{fn:starts-with}(\text{string}, \text{string})$ : boolean $\text{fn:string-length}(\text{string?})$ : number $\text{fn:substring}(\text{string}, \text{number}, \text{number?})$ : string $\text{fn:substring-after}(\text{string}, \text{string})$ : string $\text{fn:substring-before}(\text{string}, \text{string})$ : string	$/$ and eq ne lt le gt ge or $+$
<input type="checkbox"/> Show XML Schema groups <b>XPath Expression</b> <code>fn:starts-with(., 'A')</code>		

Integration Toolkit  
 DFDL Schema editor  
 contains an XPath  
 Expression Builder to  
 help you build the  
 expression

© Copyright IBM Corporation 2016

Figure 5-7. DFDL expressions

WM676/ZM6761.0

### Notes:

DFDL expressions can be used to set some properties dynamically at processing time.

The example on this figure shows the use of an expression to provide the count of the number of occurrences for an unbounded array element that is called value. The example shows the use of a relative path to refer to a count element earlier in the data.

The DFDL expression language is a subset of XPath 2.0. Expressions follow XPath 2.0 syntax rules, but are always enclosed in the brace characters “{” and “}”.

A discriminator is an example of a DFDL expression. Discriminators are described later in this unit.

## 5.2. Modeling complex data

## Understanding the logical structure of the data

### 1. Identify complex structures.

- Complex types
- Complex elements

### 2. Identify simple items.

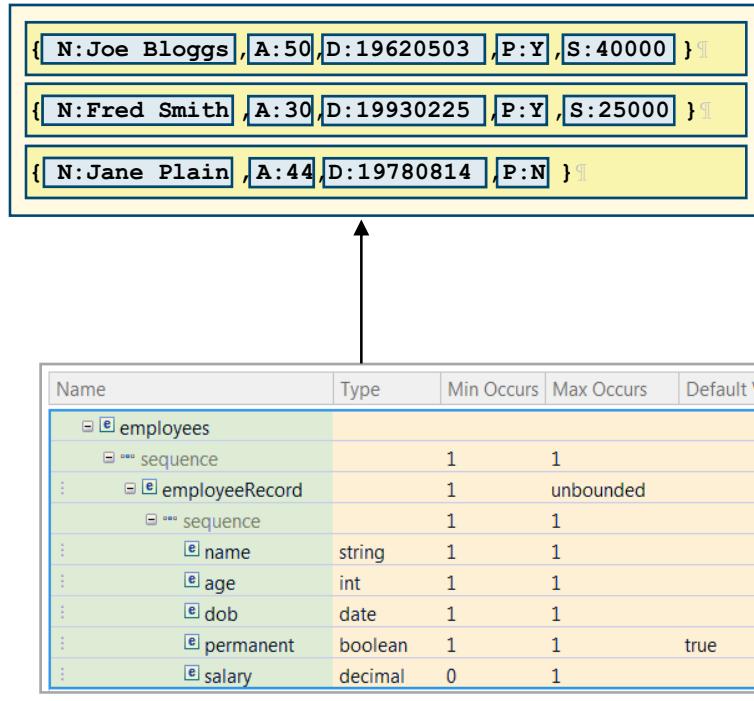
- Simple types
- Simple elements

### 3. Identify structure order.

- Sequence groups
- Choice groups
- Unordered groups

### 4. Identify cardinality with **Min Occurs** and **Max Occurs**.

### 5. Identify elements that can be nil (null) and default values.



© Copyright IBM Corporation 2016

Figure 5-8. Understanding the logical structure of the data

WM676/ZM6761.0

### Notes:

Before you can model complex data in DFDL, you must understand the logical structure and the content of the data. Understanding the logical structure of your data involves five stages.

#### 1. Identify the complex structures that correspond to the complex types in the model.

The entire data itself has one complex type. If the data contains substructures, each substructure has a complex type. For example, a file is a complex structure. It contains records, which are also complex structures. Similarly, a database table is a complex structure. A database table contains rows, which are also complex structures.

#### 2. Identify the simple items.

Simple items occur within each complex type, and each has a logical data type. Simple items correspond to simple elements. For example, each field in a COBOL copybook with a PIC clause, or each field in a file, corresponds to an element of a simple type.

#### 3. Identify the structure order rules.

This step determines whether the group within a complex type is a sequence group, an unordered group, or a choice group. In a choice group only one of the listed items can occur. Examples are C unions, COBOL REDEFINES, and files that contain mixed data.

---

4. Identify complex structure and simple item cardinality.

This step provides the values for the **minOccurs** and **maxOccurs** logical properties of the elements. Is an element required (`minOccurs != 0`) or optional (`minOccurs = 0`)? Is an element an array (`maxOccurs > 1`)? If so, is there a fixed number of occurrences (`minOccurs = maxOccurs`) or a variable number of occurrences (`minOccurs != maxOccurs`)? Can the number of occurrences be unlimited (`maxOccurs = unbounded`)?

5. Identify nillable items and default values.

It might be necessary for some elements to carry a special out-of-band value, in which case they must be nillable. For example, a numeric field in a COBOL copybook might sometimes be set to SPACES, which is not legal for a DFDL number. Some required elements might be empty in the data, in which case a default value can be provided.

The DFDL parser can detect a null value that is represented as an out-of-range value. The null value must be specified in the DFDL schema, and can be the same or different for each element. In DFDL, it is called a *nil value*. While parsing, the DFDL parser checks the nil value for each element in the message. If the value in the bitstream matches the nil value in the DFDL schema, the DFDL parser sets the value in the message tree to NULL. The same check is done when converting a message tree to a bitstream. If the value in the message tree is NULL, the DFDL parser outputs the nil value from the DFDL schema.

## Configuring the DFDL annotations (1 of 2)

- All elements
  - Delimiters: Initiator, Terminator, Encoding
  - Length establishment: Length Kind, Length
  - Number of occurrences: Occurs Count Kind, Occurs
  - Alignment rules: Alignment, Fill Byte
  - Nillable
  - Discriminator needed
- Simple elements
  - Text: Representation, Encoding, Text, Escape Scheme Ref
  - Binary: Representation, Byte Order
  - Type is string: `textString`
  - Type is number: `text Number`, `binaryNumber`
  - Type is Boolean: `textBoolean`, `binaryBoolean`
  - Type is Calendar: `calendar`, `textCalendar`, `binaryCalendar`
  - Split properties between Element and SimpleType

employeeRecord (Element)	
<type filter text>	
Property	Value
Comment	
General	
Encoding (code page)	US-ASCII
Byte Order	bigEndian
Content	
Length Kind	implicit
Occurrences	
Min Occurs	1
Max Occurs	unbounded
Occurs Count Kind	implicit
Delimiters	
Initiator	{
Terminator	}%CR;%LF;

© Copyright IBM Corporation 2016

Figure 5-9. Configuring the DFDL annotations (1 of 2)

WM676/ZM6761.0

### Notes:

After the logical structure of your data is established, the DFDL annotations (properties) are added to describe the physical format of the components.

For all elements (simple and complex), ask yourself these questions. Does the element have any delimiters, that is, an initiator or a terminator? If so, what is the encoding, and are they present when the element is empty or nil? How is the content of the element established? The answers to these questions determine the **Length Kind** property value:

- **explicit** for a fixed length
- **prefixed** if it has a length prefix
- **delimited** if bounded by a delimiter
- **pattern** to use a regular expression
- **implicit** if the types determine the length

If the element is optional or an array, then how is the number of occurrences established? Are there any alignment rules to apply? How is any nil value described? Is an assertion or discriminator needed to establish whether the element exists?

For simple elements, ask yourself these questions about the data:

- Is the element text or binary representation? This answer and its simple type determine the other properties that must be set.
- For text formats, is an escape scheme needed?

## Configuring the DFDL annotations (2 of 2)

- Sequence
  - Ordered or unordered: Sequence Kind
  - Separator: Separator, Separator Position, Separator Policy, Encoding
  - Unique initiators for all sub elements: Initiated Content
- Choice
  - Same length for all branches: Choice Kind
  - All branches have unique initiators: Initiated Content
  - Branches need discriminators?

© Copyright IBM Corporation 2016

Figure 5-10. Configuring the DFDL annotations (2 of 2)

WM676/ZM6761.0

### Notes:

For sequences, ask yourself these questions about the data:

- Is the sequence ordered or unordered?
- Does the sequence have a separator that delimits its components? If so, is the separator's position infix, prefix, or postfix, and are there any circumstances when separators are suppressed, such as when optional elements are missing?
- Do all the subelements of the sequence have unique initiators that can identify that they exist?
- Does the sequence itself have an initiator or a terminator?

For choice groups, ask yourself these questions about the data:

- Is the choice one where all the branches must occupy the same length, or not?
- Do all the branches of the choice have unique initiators that can identify which one appears?
- Are discriminators needed on the branches to establish which one appears?
- Does the choice itself have an initiator or a terminator?

## Configuring the DFDL annotations example

```
{ N:Joe Bloggs , A:50 , D:19620503 , P:Y , S:40000 }  

{ N:Fred Smith , A:30 , D:19930225 , P:Y , S:25000 }  

{ N:Jane Plain , A:44 , D:19780814 , P:N }
```

Name
<b>E employees</b>
<b>sequence</b>
<b>E employeeRecord</b>
<b>sequence</b>
<b>E name</b>
<b>E age</b>
<b>E dob</b>
<b>E permanent</b>
<b>E salary</b>

- **employees**  
Initiator = <no initiator>, Terminator = <no terminator>, Length Kind = implicit
- **employeeRecord**  
Initiator = {, Terminator = }%CR;%LF;, Encoding = ASCII, Length Kind = implicit, Occurs Count Kind = implicit
- **employeeRecord sequence**  
Sequence Kind = ordered, Separator = , , Separator Position = infix, Separator Policy = suppressedAtEnd
- **salary**  
Initiator = S:, Terminator = <no terminator>, Encoding = ASCII, Length Kind = delimited, Representation = text, Text Number Rep = standard, Text Number Pattern = #0.##
- **permanent**  
Initiator = P:, Terminator = <no terminator>, Encoding = ASCII, Length Kind = delimited, Representation = text, Text Boolean True Rep = Y, Text Boolean False Rep = N

© Copyright IBM Corporation 2016

Figure 5-11. Configuring the DFDL annotations example

WM676/ZM6761.0

### Notes:

This figure provides an example of DFDL annotation configuration. The upper portion of the figure includes the sample data and the DFDL model. The bullets identify some of the DFDL elements and their key properties.

The top-level element, named **employees**, is a complex element that defines the entire file. The file has no initiator or terminator, so those properties are not set. The **Length Kind** property is set to **implicit**, which means that its contents for this element (the records in the file) define the length of the **employees** element.

The **employeeRecord** element is a complex element that defines a single record in the file. In the example data, each record begins with a “{” character and ends with a “}” character. Each record ends with a carriage return and line feed character. In the DFDL schema editor, the **Initiator** and **Terminator** properties are set to the appropriate values. The **Length Kind** property is set to **implicit**, which means that the contents of this complex element (the fields in the record) determine the length of the **employeeRecord** element.

The **employeeRecord sequence** element defines the **employeeRecord** element as an ordered sequence, which means that contents of the **employeeRecord** (fields) must always appear in the

same order. The separator between each subelement is a comma, which is between each field (infix).

The **S:** characters prefix the **salary** field. The **Text Number Pattern** property identifies the implied format of the salary value.

The **P:** characters prefix the **permanent** field. The **Text Boolean True Rep** and **Text Boolean False Rep** identify the true and false values.

## DFDL points of uncertainty

- DFDL parser is a recursive-descent parser with look-ahead for resolving “points of uncertainty”:
  - A choice
  - An optional element
  - A variable array of elements
- DFDL parser must speculatively attempt to parse data until an object is either “*known to exist*” or “*known not to exist*”
  - Until that applies, the occurrence of a processing error causes the parser to suppress the error, backtrack, and then make another attempt
- Discriminator annotation can be used to assert that an object is “*known to exist*”, which prevents incorrect back tracking
- Initiators can also assert “*known to exist*”

© Copyright IBM Corporation 2016

Figure 5-12. DFDL points of uncertainty

WM676/ZM6761.0

### Notes:

When the DFDL encounters constructs such as a choice, an optional element, or a variable array, it might not be able to determine whether it can parse those constructs. The parser tries to parse the data until it can determine for certain whether an object *exists* or *does not exist*. When the parser cannot determine either of those states, it can cause the parser to backtrack and try parsing the object by a different means. This backtracking increases the expense of the parsing operation.

To reduce the amount of backtracking, you can add annotations or initiators to the DFDL model as a “hint” to the parser.

## DFDL points of uncertainty example

```

<xs:choice>
  <xs:element name="Update" >
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Type" type="xs:int" dfdl:representation="binary"
      ...>
        <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/" >
          <dfdl:discriminator test=". eq 1" />
        </xs:appinfo></xs:annotation>
      </xs:element>
      ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Create" >
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Type" type="xs:int" dfdl:representation="binary"
      ...>
        <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/" >
          <dfdl:discriminator test=". eq 2" />
        </xs:appinfo></xs:annotation>
      </xs:element>
      ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:choice>

```

Discriminator  
resolves the choice

© Copyright IBM Corporation 2016

Figure 5-13. DFDL points of uncertainty example

WM676/ZM6761.0

### Notes:

This figure is an example of a schema that uses a discriminator to resolve the point of uncertainty, which is a choice.

In this example, the schema path for the element that is named **Update** is chosen if the **Type** element equals 1. If the **Type** element equals 2, the schema path for the element that is named **Create** is chosen.



## Adding a discriminator to a DFDL element

1. In the DFDL Schema editor, select the DFDL schema element on which you want to add a discriminator.
2. On the **Asserts and Discriminators** tab of the DFDL Properties area, click **Discriminator**.
3. Click **Add discriminator**.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field.
  - Content assistance is available for the **Test Condition** field
  - Type Ctrl+Space to open the Path Expression Builder

The screenshot shows the 'Asserts and Discriminators' tab selected in the DFDL Schema editor. The 'Asserts' section is visible above, and the 'Discriminator' section is highlighted with a red box. The 'Discriminator' section contains a table with three columns: 'Test Kind', 'Test Condition', and 'Message'. A new row is being added, indicated by the 'Add discriminator' link in the 'Test Condition' column.

© Copyright IBM Corporation 2016

Figure 5-14. Adding a discriminator to a DFDL element

WM676/ZM6761.0

### Notes:

You can add a discriminator to a schema object on the **Asserts and Discriminators** tab in the Integration Toolkit DFDL schema editor.

To add a discriminator to an element:

1. Select the DFDL schema element.
2. On the **Asserts and Discriminators** tab of the DFDL properties area, click **Discriminators**.
3. Click the **Add discriminator** link or the green plus sign.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field. Content assistance is available for the **Test Condition** field; press Ctrl+Space to open the XPath Expression Builder.

## Adding asserts

- Assert annotation element is used to assert truths about a DFDL model that are used only when parsing to ensure that the data is well-formed
- Evaluates a Boolean expression
- Checks are separate from validation checking and are done even when validation is off

### Examples:

- Test that an order quantity field is not zero
- Test that the string for purchase order starts with the letter “A”

© Copyright IBM Corporation 2016

Figure 5-15. Adding asserts

WM676/ZM6761.0

### Notes:

You can add asserts to a DFDL schema object to define tests that ensure that the data is well-formed.

In the examples on the figure, the first assert tests the data to ensure that a value is not zero. The second assert tests for a precondition violation. The third assert contains a DFDL expression that evaluates to true or false. If the expression evaluates to true, parsing continues. If the expression evaluates to false, a processing error is raised.



## Adding an assert to DFDL element

1. In the DFDL Schema editor, select the DFDL schema element on which you want to add an assert.
2. On the **Asserts and Discriminators** tab of the DFDL **Properties** area, click **Asserts**.
3. Click **Add assert**.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field.
  - Content assistance is available for the **Test Condition** field
  - Type Ctrl+Space to open the Path Expression Builder

The screenshot shows the DFDL Schema editor interface. At the top, there are tabs: 'Representation Properties', 'Variables', and 'Asserts and Discriminators'. The 'Asserts and Discriminators' tab is selected, showing a sub-tab 'sequence'.

**Asserts Tab:**

- Header:** Assert defines a test to be used to ensure the data are well formed. Assert is used only when parsing data. Only asserts with test expressions are supported in the current IBM DFDL implementation.
- Table:** Test Kind | Test Condition | Message
- Buttons:** Add assert (+) and Delete (x).

**Discriminator Tab:**

- Header:** Discriminator defines a test to be used when resolving a point of uncertainty such as choice branches or optional elements. Discriminator is used only when parsing data to resolve the point of uncertainty to one of the alternatives. Only discriminators with test expressions are supported in the current IBM DFDL implementation.
- Table:** Test Kind | Test Condition | Message
- Buttons:** Add discriminator (+) and Delete (x).

© Copyright IBM Corporation 2016

Figure 5-16. Adding an assert to DFDL element

WM676/ZM6761.0

### Notes:

To add an assert in the DFDL schema editor:

1. Select the DFDL schema element.
2. On the **Asserts and Discriminators** tab of the DFDL properties, click **Asserts**.
3. Click **Add assert**.
4. Enter the test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field. Content assistance is available for the **Test Condition** field; press Ctrl+Space to open the XPath Expression Builder.

## Asserts with recoverable errors

- If the **Test Condition** expression evaluates to “false”, the **Failure Type** property determines the error type:
  - Processing error** generates an exception and stops processing the data
  - Recoverable error** logs the error and continues processing the data
- In a message flow, DFDL recoverable errors are handled like runtime validation errors but are generated regardless of whether runtime validation is enabled

The screenshot shows the IBM Integration Bus Studio interface. On the left, there is a tree view of a schema definition with nodes like 'Unordered', 'Type1', 'A1', 'B1', 'C1', and 'Type2'. On the right, there is a panel titled 'Asserts' with the following content:

Test Kind	Test Condition	Message	Failure Type
expression	{fn:contains (,'a')}	No 'a'	recoverableError
Add assert			

A red box highlights the 'Failure Type' column header and the 'recoverableError' entry.

© Copyright IBM Corporation 2016

Figure 5-17. Asserts with recoverable errors

WM676/ZM6761.0

### Notes:

If you add an assert, you can specify the action if the assert identifies invalid data.

Selecting the recoverable error option is useful if you need to check the physical data instead of the logical structure of the data. For example, you might want to check the text length of a number or calendar data type but you do not want to stop the parser if the text length is not within the expected range.

## Length prefixes

- When a prefix to the element contains the length of the element itself

Example: Addr : 158200 Warden Ave 12Markham, Ont 07L3G 1H7

The diagram shows the string "Addr : 158200 Warden Ave 12Markham, Ont 07L3G 1H7" enclosed in a box. Below the box, four arrows point upwards to the numbers 15, 12, and 7, which are labeled "15 bytes", "12 bytes", and "7 bytes" respectively. The first two digits "15" represent the length of the "Warden" field, the next two digits "12" represent the length of the "Markham" field, and the final three digits "07L3G" represent the length of the postal code field.

- Common variations

- Value in the length prefix represents the length of the element to which it refers
- Value in the length prefix includes the length of the prefix and that of the element
- Length prefix has different characteristics from the element

© Copyright IBM Corporation 2016

Figure 5-18. Length prefixes

WM676/ZM6761.0

### Notes:

Some data might contain a *length prefix*. A length prefix is a portion of the data that identifies the number of bytes in a simple element.

In the **Addr** element in the example on the figure, the first 2 characters (15) are the number of bytes of data in the **Addr** element. The next field begins after the 15 bytes of data. In the example, the next field has a length prefix of 12, which means that the next 12 bytes are data.

The value in the length prefix might represent the length of the element to which it refers, or the value in the length prefix might include the length of the prefix and that of the element. The length prefix might have different characteristics from the element; for example, it might be a binary prefix whereas the element is text. It is even possible for a length prefix to have another length prefix provide its own length.

## Length prefixes example

Addr : 158200 Warden Ave 12 Markham, Ont 07L3G 1H7

1. Define simple type element that is named **TwoCharsText** with physical characteristics of the prefix.

<b>Content</b>	<b>short</b>
Representation	
Length Kind	explicit
Length	2
Length Units	
<b>Text Content</b>	
Text Number Representation	
Number Pattern	00

2. Modify the **Content** elements to refer to the new simple type element.

<b>Content</b>	<b>string</b>
Representation	
Length Kind	prefixed
Length Units	
Prefix Length Type	TwoCharsText
Prefix Includes Prefix Length	no

Name of element that is defined in Step 1

© Copyright IBM Corporation 2016

Figure 5-19. Length prefixes example

WM676/ZM6761.0

### Notes:

The figure shows a technique for defining length prefixes in your DFDL model.

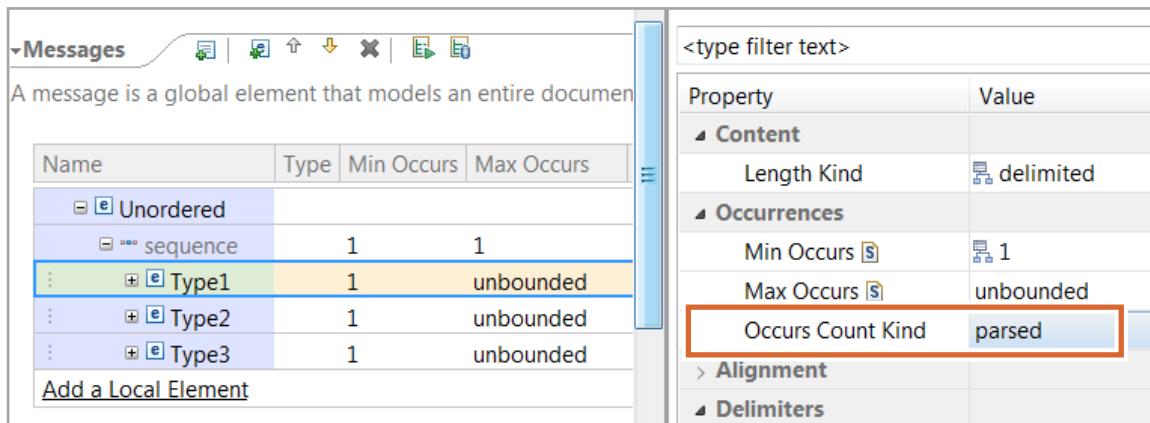
First, create a simple type that represents the prefix data. In the example, the simple type that represents the prefix is named **TwoCharsText**.

Then, for any elements that have a prefix, modify the **Content** elements:

- Set the **Length Kind** to: prefixed
- Set the **Length Units** to: characters
- Set the **Prefix Length Type** to the simple type that represents the prefix data.

## Parsed arrays

- DFDL parser determines the number of occurrences by parsing the data instead of referencing **minOccurs** and **maxOccurs**
  - Specified by setting **Occurs Count Kind** to **parsed**
  - Can specify on optional elements and array elements



© Copyright IBM Corporation 2016

Figure 5-20. Parsed arrays

WM676/ZM6761.0

### Notes:

The element's **Min Occurs** and **Max Occurs** properties typically determine the allowed number of occurrences of an element in the data stream. An element with a **Max Occurs** value that is greater than 1 or set to **unbounded** is an array.

In DFDL, you can specify how you want the DFDL parser to determine the number of occurrences of an element. If the number of occurrences varies and the parser should determine the number automatically, set the **Occurs Count Kind** property to **parsed** – for example, if the occurrences are determined by using an initiator or a separator.

## Direct dispatch choice

- Identifies choice branch to take during parsing
  - DFDL parser evaluates the expression and jumps to the indicated branch, without speculative parsing
- Example: An element in the data identifies the choice branch
- Specified by setting:
  - Choice Dispatch Key** on **choice** element to a string
  - Choice Branch Key** on each element in the **choice** group to a string that uniquely identifies that branch

choice			
<type filter text>			
Property		Value	
Content		Initiated Content	no
		Choice Length Kind	implicit
		Choice Dispatch Key	{./Header/Tag}
Occurrences		Min Occurs	1
		Max Occurs	1

© Copyright IBM Corporation 2016

Figure 5-21. Direct dispatch choice

WM676/ZM6761.0

### Notes:

In some data that contains a choice group, an earlier field in the data stream indicates the branch to take in the logical data model. For example, your data model might support files that can contain both purchase order and invoice records. A field in the data stream might identify whether this data stream is for a purchase order or for an invoice.

If the data contains some data that identifies the choice branch to take in the DFDL model, the **Choice Dispatch Key** property of the choice can be used to look at the data. It then can match it to the **Choice Branch Key** property of one of the branches. If a match is found, then that choice branch is deemed to be present. If a subsequent component in that choice branch fails to parse, it is a real parsing error in that choice branch and no backtracking takes place.

## Unordered sequences

- Elements can occur in the data in any order, and are arranged into schema order in the tree by the DFDL parser
- Specified by setting **Sequence Kind** to **unordered**
  - Only elements are allowed in the sequence
  - Must have unique name or namespace
  - If optional or array element, **Occurs Count Kind** must be set to **parsed**
- Order of the data is not preserved in the message tree
- Data in the message tree must be in schema order before serializing

Name	Type	Min Occurs	Max Occurs
Unordered			
sequence		1	1
Type1		1	unbounded
Type2		1	unbounded
Type3		1	unbounded

**Content**

Initiated Content	yes
Sequence Kind	unordered

**Occurrences**

Min Occurs	1
Max Occurs	1

© Copyright IBM Corporation 2016

Figure 5-22. Unordered sequences

WM676/ZM6761.0

### Notes:

In a DFDL unordered sequence, elements can occur in the data in any order. A DFDL unordered sequence is specified by setting the **Sequence Kind** property to **unordered** on the **sequence** element.

If you identify a **sequence** element as **unordered**, some restrictions apply to the content of the **sequence** element:

- Only elements are allowed in the **sequence**.
- Each element must have a unique name-namespace combination.
- If the element is optional or is an array, **Occurs Count Kind** must be set to **parsed**.

The order in the data is not preserved in the DFDL logical model; the model is static. After parsing, the data is arranged in the order that is declared in the DFDL schema.

## Adding a reference to another schema (1 of 3)

- Create reference to another DFDL schema in the workspace when one schema file contains DFDL objects that you want to refer to in another schema
- Namespaces of the two schema files determine whether to use the **import** or **include** option

	Target file has a target namespace	Target file has no target namespace
Parent file has target namespace	<b>xsd:import</b>	<b>xsd:include</b>
Parent file has no target namespace	<b>xsd:import</b>	<b>xsd:include</b>

© Copyright IBM Corporation 2016

Figure 5-23. Adding a reference to another schema (1 of 3)

WM676/ZM6761.0

### Notes:

A DFDL schema can reuse message objects from another DFDL schema file in the workspace.

Two mechanisms are available for reusing message definition files: `import` and `include`. As shown in the table, the namespace of the schemas determines whether the `import` or `include` command is used.

When a target namespace file includes a “no target” namespace file, referencing an object in the target file from the parent file causes the object to be present in the namespace of the parent file.

When `import` or `include` is used, global objects from the target file can be used in the parent file.

The namespace of objects in the target file is preserved in the parent file, with the exception noted in the previous table of a target namespace file that includes a “no target” namespace file.

## Adding a reference to another schema (2 of 3)

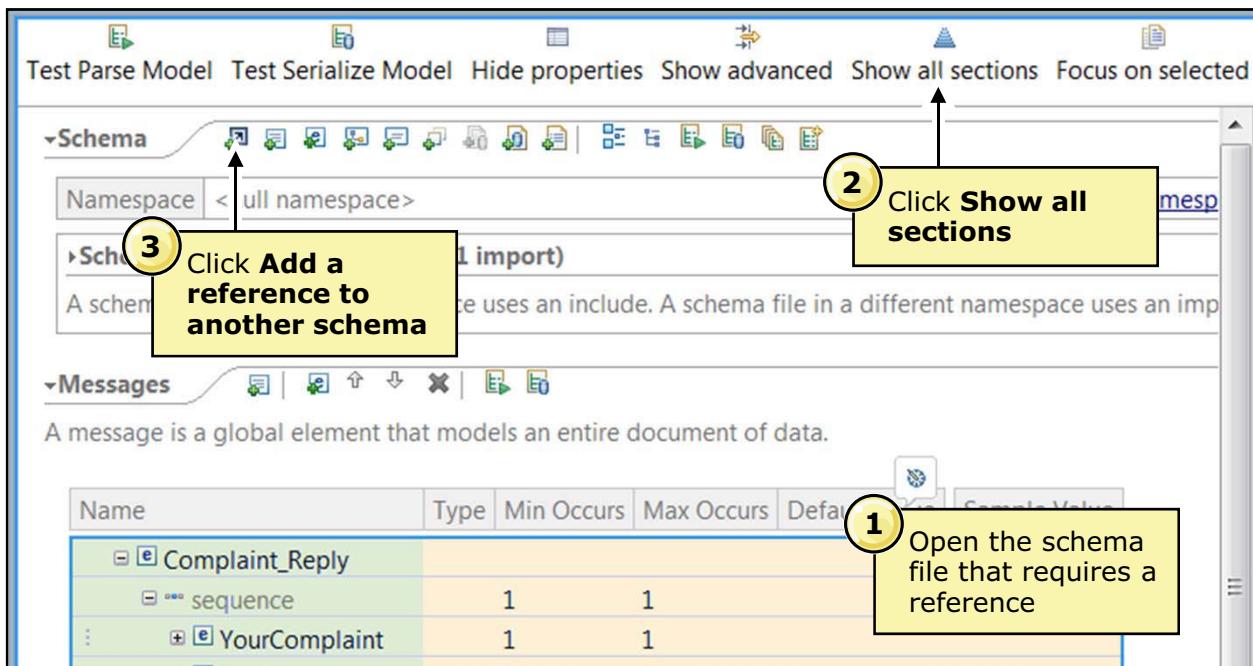


Figure 5-24. Adding a reference to another schema (2 of 3)

WM676/ZM6761.0

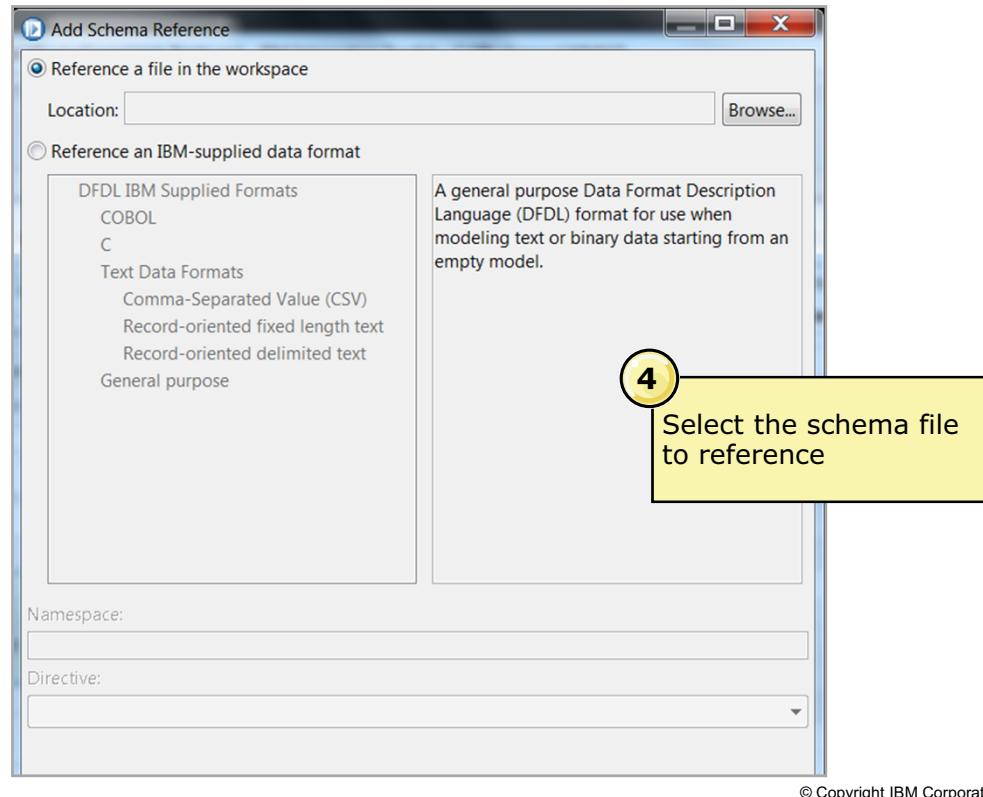
### Notes:

To add a reference to a schema:

1. Open the DFDL schema file that requires a reference in the DFDL schema editor.
2. Click **Show all sections** in the DFDL schema editor toolbar to show the **Schema** section in the DFDL schema editor view.
3. In the **Schema** section, click the **Add a reference to another schema** icon.



## Adding a reference to another schema (3 of 3)



© Copyright IBM Corporation 2016

Figure 5-25. Adding a reference to another schema (3 of 3)

WM676/ZM6761.0

### Notes:

4. Select the schema file to reference from the workspace. As an option you can reference an IBM-supplied schema.

## 5.3. Industry-standard models

If you are working with industry-standard data, a DFDL schema that defines the data might exist. The next topic provides an overview of some of the industry-standard DFDL schemas that are available in IBM Integration Bus add-on packs and on the GitHub website.

## DFDL schemas for industry formats

- ISO8583
- NACHA
- UN/EDIFACT and EDI X12
- HL7
- SWIFT MT
- Transaction log (TLOG) emitted by IBM/Toshiba 4690 point-of-sale devices

Many DFDL schemas are available on the GitHub DFDL Schemas for Commercial and Scientific Data Formats website: <https://github.com/DFDLSchemas>

© Copyright IBM Corporation 2016

Figure 5-26. DFDL schemas for industry formats

WM676/ZM6761.0

### Notes:

If you are working with industry-standard data, a DFDL schema that defines the data might already be available. The figure lists the DFDL schemas that are available for industry-standard formats.

- ISO 8583 is the International Organization for Standardization (ISO) standard for systems that exchange cards.
- NACHA manages the development, administration, and governance of the backbone for the electronic movement of money and data in the United States.
- EDIFACT (ISO 9735) is the international standard for electronic data interchange (EDI). The term stands for Electronic Data Interchange For Administration, Commerce, and Transport
- HL7 is a standard for exchanging information between medical applications. The DFDL schema for HL7 is available in the IBM Integration Bus Healthcare Pack.
- IBM/Toshiba 4690 SurePOS ACE is point-of-sale application. The DFDL schema is available on GitHub.

GitHub is a web-based hosting service for software development projects that use the Git revision control system.



# DFDL for ISO 8583

- Text/binary format for ATM and credit card transactions
    - Message consists of a flat structure of simple data fields
    - Data fields are either fixed length or variable length with a prefix
  - Most data fields are optional (Min Occurs “0”) but there are no delimiters
  - A flag in a special bitmap indicates the presence of a field in the data
    - **Occurs Count Kind** = expression
    - **Occurs Count** = { /ISO8583 1987/PrimaryBitmap/Bitxxx }

Name	Type	Min Occurs	Max Occurs
ISO8583_1987			
sequence		1	1
MTI_Version	integer	1	1
MTI_MessageClass	integer	1	1
MTI_MessageFunction	integer	1	1
MTI_MessageOrigin	integer	1	1
Bitmaps_Group		1	1
sequence		1	1
PrimaryBitmap	PrimaryBitmapType	1	1
SecondaryBitmap	SecondaryBitmapType	0	1
<Type_n_LL>		0	1
ProcessingCode_003	<Type_n_string>	0	1
AmountTransaction_004	<Type_n_decimal>	0	1
AmountSettler_005	<Type_n_decimal>	0	1
AmountCardHolderBilling_006	<Type_n_decimal>	0	1
TransmissionDateTime_007	<Type_n_dateTime>	0	1
AmountCardHolderBillingFee_008	<Type_n_decimal>	0	1

Figure 5.37 DEPI for ISO 8583

---

WMGZ6/ZMGZ61.0

#### **Notes-**

ISO 8583 is a standard for systems that exchange cards. It has three parts:

- Messages, data elements, and code values
  - Application and registration procedures for Institution Identification Codes (IIC)
  - Maintenance procedures for messages values

The binary representation of a message is as follows:

08002020000008000000000000000000013239313130303031

In the DFDL schema for ISO8583, most data fields are defined as optional but they have no delimiters. So the DFDL schema contains two simple types (**PrimaryBitmap** and

**SecondaryBitmap**) that define the special bitmap that indicates the presence of a field in the data. Any elements that must use the special bitmap have the **Occurs Count Kind** and **Occurs Count** properties that are set to reference the bitmap types.

## DFDL for NACHA

- NACHA manages the development, administration, and governance of the backbone for the electronic movement of money and data in the United States
- Different kinds of records exist, but only one kind appears in a specific batch
  - Use a choice with a discriminator on each branch
- All records are 94 characters long and usually terminated with a new line

Name	Type	Min Occurs	Max Occurs
NACHAFile		1	1
sequence		1	1
FileHeaderRecord		1	1
Batch	sequence	1	unbounded
BatchHeaderRecord		1	1
EntrvDetail	choice	1	1
CCDEntryDetailRecord		1	1
PPDEntryDetailRecord		1	1
TELEntryDetailRecord		1	1
WEBEntryDetailRecord		1	1
CTXEntryDetailRecord		1	1
BatchControlRecord		1	1
FileControlRecord		1	1

<type filter text>

Property	Value
General	
Data Format Reference	nch:RecordWithT
Encoding (code page)	US-ASCII
Byte Order	bigEndian
Ignore Case	no
Fill Byte	%SP;
Content	
Length Kind	explicit
Length	94
Length Units	bytes
Occurrences	
Min Occurs	1
Max Occurs	1

© Copyright IBM Corporation 2016

Figure 5-28. DFDL for NACHA

WM676/ZM6761.0

### Notes:

The DFDL schema for NACHA data supports record types CCD, CTX, ACK, and ATX.

## DFDL for EDIFACT

- EDIFACT is the international standard for electronic data interchange (EDI)
- Schema models the entire EDIFACT interchange
- For parsing or serializing only, no validation rules
- Use 1 GB JVM heap size when deploying and for the Integration Toolkit

```

UNB+UNOA:1+005435656:1+006415160:1+
060515:1434+00000000000778'
UNH+00000000000117+INVOIC:D:97B:UN'
BGM+380+342459+9'
DTM+3:20060515:102'
RFF+ON:521052'
NAD+BY+792820524::16++CUMMINS MID-
RANGE ENGINE PLANT'
NAD+SE+005435656::16++GENERAL
WIDGET COMPANY'
CUX+1:USD'
LIN+1++157870:IN'
IMD+F++:::WIDGET'
QTY+47:1020:EA'
ALI+US'
MOA+203:1202.58'
PRI+INV:1.179'
LIN+2++157871:IN'
IMD+F++:::DIFFERENT WIDGET'
QTY+47:20:EA'
ALI+JP'
MOA+203:410'
PRI+INV:20.5'
. . .

```

**SAMPLE**

© Copyright IBM Corporation 2016

Figure 5-29. DFDL for EDIFACT

WM676/ZM6761.0

### Notes:

The DFDL schema for EDIFACT models the entire EDIFACT interchange.

EDIFACT files can be large and the DFDL model is complex. It is suggested that you set the JVM heap size to a minimum of 1 GB when deploying message flow applications that process EDIFACT data.

## DFDL for HL7 v2

- Delimited text format that is used in the healthcare industry
  - Message consists of an MSH segment followed by a number of other segments
  - A three-character tag identifies each segment, which is terminated with CR
  - Segments contain variable length fields that are terminated with a delimiter
  - Fields can be simple or complex, and each level of nesting has its own delimiter
  - Fields can repeat, and occurrences have their own delimiter (“~”)
  - Delimiters are dynamically defined in the first (MSH) segment

The screenshot shows the DFDL schema for HL7 v2. On the left is a tree view of the schema structure:

Name	Type	Min Occurs	Max Occurs
HL7		1	1
sequence		1	1
MSH	MSH.CONTENT	1	1
sequence		1	1
MSH.1.FieldSeparator	string	1	1
MSH.2.ServiceString	MSH.2.ServiceString.CONTENT	1	1
sequence		1	1
anyHL7Segment	anyHL7Segment.TYPE	0	unbounded
choice		1	1
choice		1	1
SFT	SFT.CONTENT	1	1
MSA	MSA.CONTENT	1	1
ARQ	ARQ.CONTENT	1	1

On the right is a properties panel:

<type filter text>									
Property	Value								
Comment	\$								
General									
Content	<table border="1"> <tr> <td>Length Kind</td> <td>delimited</td> </tr> </table>	Length Kind	delimited						
Length Kind	delimited								
Occurrences									
Alignment									
Delimiters	<table border="1"> <tr> <td>Initiator</td> <td>MSH</td> </tr> <tr> <td>Terminator</td> <td>%NL;</td> </tr> <tr> <td>Empty Value Delimiter</td> <td>both</td> </tr> <tr> <td>Output New Line</td> <td>%CR;</td> </tr> </table>	Initiator	MSH	Terminator	%NL;	Empty Value Delimiter	both	Output New Line	%CR;
Initiator	MSH								
Terminator	%NL;								
Empty Value Delimiter	both								
Output New Line	%CR;								

© Copyright IBM Corporation 2016

Figure 5-30. DFDL for HL7 v2

WM676/ZM6761.0

### Notes:

HL7 data is delimited text. Each message consists of an MSH segment and other segments. The following example shows some HL7 data:

```
MSH|^~\&|CERNER||PriorityHealth|||ORU^R01|Q479004375T431430612|P|2.3|
PID|||001677980||SMITH^CURTIS||19680219|M|||||||929645156318|123456789|
PD1|||1234567890^LAST^FIRST^M^^^^^NPI|
OBR|1|341856649^HNAM_ORDERID|000002006326002362|648088^Basic Metabolic
Panel|||20061122151600|||||||1620^Hooker^Robert^L|||||20061122154733|||F|||||
|||||20061122140000|
OBX|1|NM|GLU^Glucose Lvl|59|mg/dL|65-99^65^99|L|||F|||20061122154733|
```

The figure shows part of the DFDL schema for HL7. In the schema, a **choice** group and the **Initiator** property value determine the segment.

## DFDL for TLOG

- A “transaction log” consists of multiple different transaction records
- Each transaction record has a type (and some records have a subtype)
  - Use a choice with a discriminator on each branch
- Each transaction record is a sequence of delimited binary fields
- Most of the fields are a special packed decimal unique to 4690

The screenshot shows the IBM Integration Bus Designer interface. On the left, there's a tree view of the DFDL schema. A red box highlights a choice group under TransactionRecord. A yellow box highlights a note: "Fields are defined as 4690 packed decimal". On the right, there's a properties panel with a red box around the "Content" section, which includes "Representation: binary" and "Length Kind: delimited". Another red box highlights the "Binary Content" section, specifically "Binary Number Rep: ibm4690Packed".

Name	Type	Min Occurs	Max
Transaction		1	1
sequence		1	unbo
TransactionRecord		1	1
choice		1	1
TransactionRecord00		1	1
TransactionRecord01		1	1
TransactionRecord02		1	1
sequence			
MultiPricingGroup	type_PD		
DealQuantity	type_PD	0	1
PricingMethod	type_PD	0	1
SaleQuantity	type_PD	0	1

© Copyright IBM Corporation 2016

Figure 5-31. DFDL for TLOG

WM676/ZM6761.0

### Notes:

The transaction logs that the 4690 emits are in a compact, ASCII delimited binary format, which is known as TLOG. Each TLOG consists of a number of records, called strings. Each string consists of fields that have a colon delimiter, although some strings use double quotation mark, colon, double quotation mark instead. The fields can be fixed length or variable length but the delimiter is always present. The fields can have a number of physical representations but the most frequent representation is a packed number representation.

The figure shows a portion of the DFDL schema. The schema includes a **choice** group for handling the different transaction types.

This DFDL schema is available on the GitHub website and in the IBM Integration Bus Retail Pack.



## Useful DFDL links

- OGF DFDL home page: <http://www.ogf.org/dfdl/>
- DFDL 1.0 specification:  
<http://www.ogf.org/documents/GFD.174.pdf>
- DFDL tutorials:  
[http://redmine.ogf.org/dmsf/dfdl-wg?folder\\_id=5485](http://redmine.ogf.org/dmsf/dfdl-wg?folder_id=5485)
- DFDL-WG Redmine project:  
<http://redmine.ogf.org/projects/dfdl-wg>
- DFDL Wikipedia page: <http://en.wikipedia.org/wiki/DFDL>
- DFDL Schemas on GitHub: <https://github.com/DFDLSchemas>
- Daffodil open source project:  
<https://opensource.ncsa.illinois.edu/confluence/display/DFDL/Daffodil%3A+Open+Source+DFDL>

© Copyright IBM Corporation 2016

Figure 5-32. Useful DFDL links

WM676/ZM6761.0

### Notes:

This figure lists some websites that contain useful information about DFDL and DFDL models.

## Unit summary

Having completed this unit, you should be able to:

- Reuse a DFDL model to create a more complex model
- Define length prefixes in a DFDL model
- Define a DFDL model to make parsing decisions that are based on the content of other elements in a message

© Copyright IBM Corporation 2016

Figure 5-33. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or false: DFDL is primarily designed to model data other than XML.
2. True or false: DFDL has no default properties; you must provide values for objects that need properties.
3. True or false: You must enable parse tracing when debugging a parse failure during testing.

© Copyright IBM Corporation 2016

---

Figure 5-34. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.



## Checkpoint answers

1. True or false: DFDL is primarily designed to model data other than XML.

Answer: **True**

2. True or false: DFDL has no default properties; you must provide values for objects that need properties.

Answer: **True**. You can add default values for properties in the DFDL Schema editor.

3. True or false: You must enable parse tracing when debugging a parse failure during testing.

Answer: **False**. Parse tracing is enabled by default.

© Copyright IBM Corporation 2016

Figure 5-35. Checkpoint answers

WM676/ZM6761.0

### Notes:

## Exercise 2



### Extending a DFDL model

Figure 5-36. Exercise 2

WM676/ZM6761.0

10.1

#### Notes:

In this exercise, you model complex data in DFDL. Complex data includes data with multiple record types, choice groups, and optional components. You also learn how to use discriminators in a DFDL model to optimize data parsing.

## Exercise objectives

After completing this exercise, you should be able to:

- Extend a DFDL model to add header and trailer records
- Reference a DFDL model in a new DFDL model
- Use discriminators in a DFDL model so that the parser can conditionally process elements as determined by the content of other elements in the data and optimize the parsing of data

© Copyright IBM Corporation 2016

Figure 5-37. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.



# Unit 6. Working with message sets and the MRM domain

## What this unit is about

A message set is the original container for message models that are used by WebSphere Message Broker. In IBM Integration Bus, DFDL schema files that are contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required when you use the MRM or IDOC domains. In this unit, you learn how to create a message set to define a message and use the MRM domain.

## What you should be able to do

After completing this unit, you should be able to:

- Describe the physical message formats that can be defined with an MRM message set
- Configure the logical and physical message properties
- Reference an MRM model in ESQL

## How you will check your progress

- Checkpoint
- Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Describe the physical message formats that can be defined with an MRM message set
- Configure the logical and physical message properties
- Reference an MRM model in ESQL

© Copyright IBM Corporation 2016

---

Figure 6-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- The MRM message set
- Referencing the MRM message in a message flow

© Copyright IBM Corporation 2016

Figure 6-2. Topics

WM676/ZM6761.0

## Notes:



## 6.1. The MRM message set

## MRM and message set development

- MRM domain can be used to parse and write a wide variety of message formats
  - Primarily intended for non-XML message formats
- Use a *message set* to create a model for a particular message format
- You can import message flows containing message sets from WebSphere Message Broker Version 7.0 and later into Integration Bus
  - Existing message sets can be viewed, compiled, and deployed
  - By default, imported message sets are read-only
- To modify existing message sets, or create new message sets or message definition files, enable message set development in the Integration Toolkit

**Note:** In Integration Bus, DFDL message model schema files are the preferred way to model messages for most data formats.

© Copyright IBM Corporation 2016

Figure 6-3. MRM and message set development

WM676/ZM6761.0

### Notes:

The IBM Integration Toolkit provides a central point to define message models for the MRM domain.

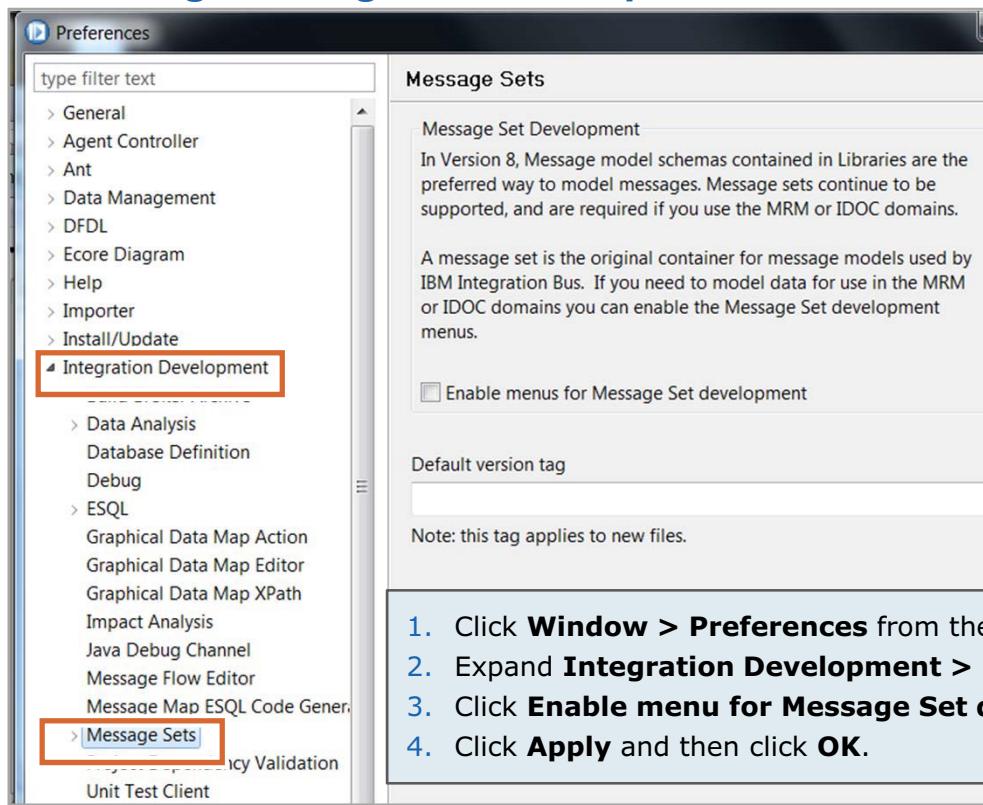
The message models that are defined for the MRM domain are platform-independent and language-independent. Through more physical format layers, a logical message structure can be represented in various physical wire formats. By allowing for importing, some of the workload of manually entering information is alleviated.

A *message set* is a logical grouping of messages and the objects that comprise them. After you create a message set, you would typically import application message formats to create and populate message definition files (.mxsd file).

After the message definition files are complete, you can then generate the message set in a form that is usable by an application, such as a WSDL or XML schema.



## Enabling message set development



© Copyright IBM Corporation 2016

Figure 6-4. Enabling message set development

WM676/ZM6761.0

### Notes:

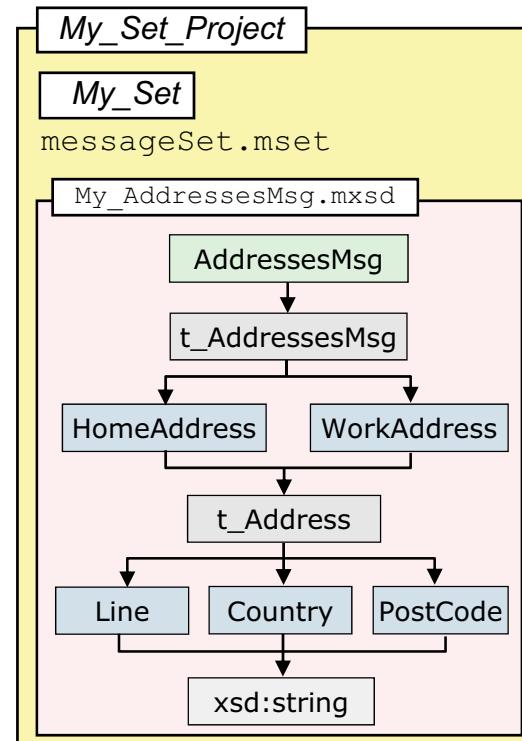
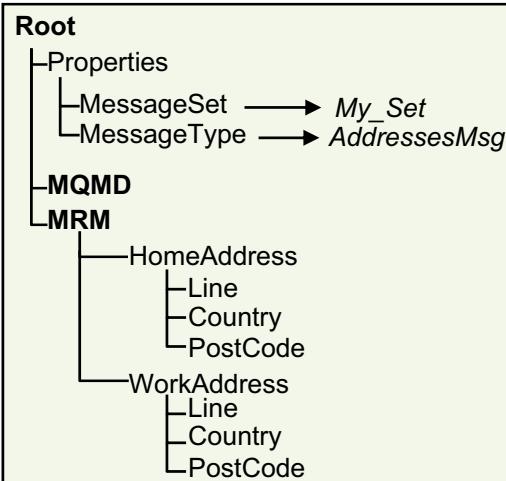
Message set development must be enabled in the Integration Toolkit **Preferences**.

## Specification, model, logical message tree

```

01 AddressesMsg.
  10 HomeAddress.
    20 Line PIC X(20) OCCURS 2 TIMES.
    20 Country PIC X(2).
    20 PostCode PIC X(10).
  10 WorkAddress.
    20 Line PIC X(20) OCCURS 2 TIMES.
    20 Country PIC X(2).
    20 PostCode PIC X(10).

```



© Copyright IBM Corporation 2016

Figure 6-5. Specification, model, logical message tree

WM676/ZM6761.0

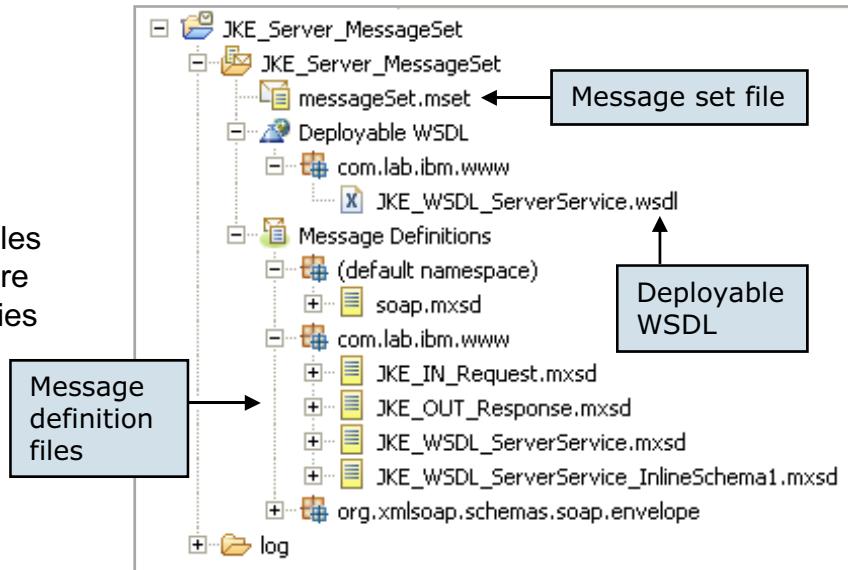
### Notes:

The sample in the figure shows the specification, structure, and the resulting logical message tree for a sample AddressesMsg.

The elements and attributes can be accessed as fields in the message body, whereas each complex type is a level in the logical message tree. For example, WorkAddress is a complex type that contains Line, Country, and PostCode.

## Message set project

- Container for resources that are associated with exactly one message set
- Logical grouping of messages and objects that comprise them, contains:
  - Message set file: Message model information common across all messages in set
  - Message definition files define logical structure and physical properties of the data
  - Deployable WSDL used by SOAP domain



© Copyright IBM Corporation 2016

Figure 6-6. Message set project

WM676/ZM6761.0

### Notes:

A message set project is a specialized project (container) in which you create and maintain all the resources that are associated with exactly one message set. A message set is a logical grouping of your messages and the objects that comprise them (elements, types, groups). The content of a message set is exactly one message set file, zero or more message definition files, WSDL files, or message category files.

The message set file provides message model information that is common across all the messages in the message set. You can create this information by using the message set editor.

When your message definition files are complete, you then generate the message set in a form that an application can use.

You can optionally group messages into message categories (files with extension.category) for convenience. You can add messages to message categories by using the message category editor. In previous versions of the product, message categories were necessary to create WSDL.

Each time that you save a message set file, message definition file, or message category file, the content is validated to ensure that the message model you are creating follows certain rules.

## Creating a message definition file

- The message definition file contains the logical structure and physical properties of the objects in XML schema form
- Create a message definition file before you create the message model objects
  - Create the message definition file from scratch
  - Base the new message definition file on an existing resource
  - Copy a message definition file from another message set

© Copyright IBM Corporation 2016

Figure 6-7. Creating a message definition file

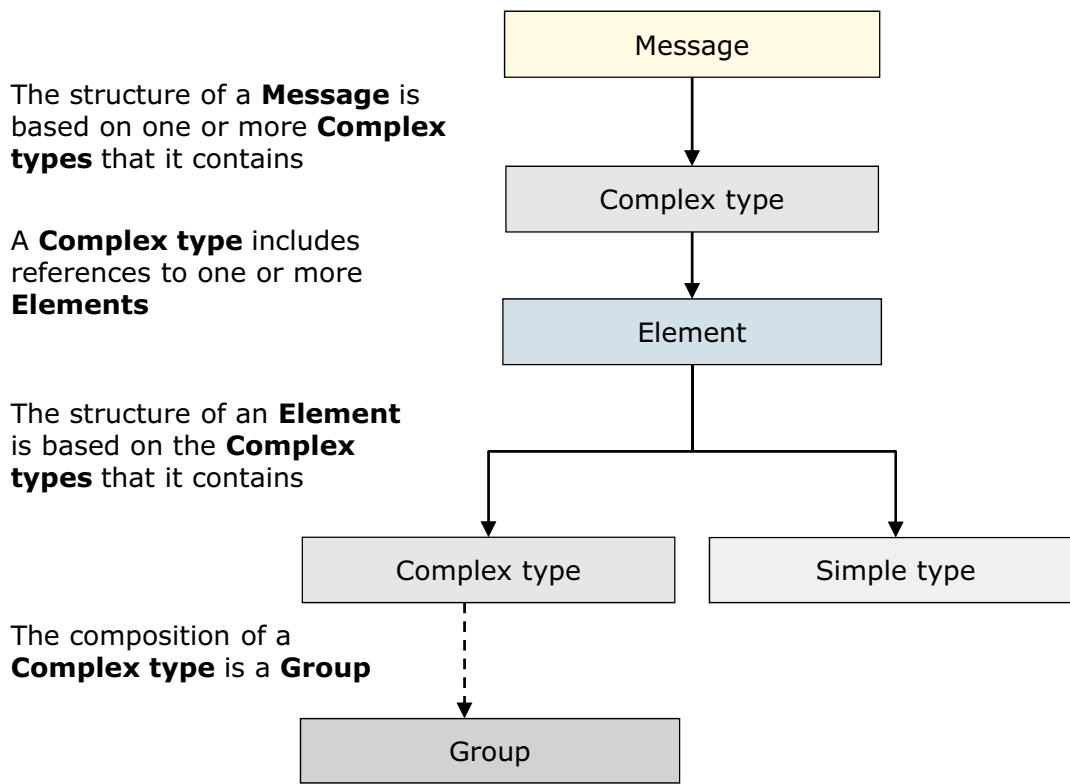
WM676/ZM6761.0

### Notes:

After you create a message set, you typically import application message formats to create and populate message definition files (.mxml). You can then edit the logical structure of your messages in the Message Definition editor.

In the Message Definition editor, you can create and edit binary, XML, and tagged-delimited physical formats that describe the precise appearance of your message bitstream during transmission. You can also create an empty message definition file and create your messages by using only the editor.

## Message definition content



© Copyright IBM Corporation 2016

Figure 6-8. Message definition content

WM676/ZM6761.0

### Notes:

A message definition is a logical description of a message. It is a structured collection of elements. The structure begins with a message, which is a set of data that is passed between applications. Messages must have a structure and format that is agreed upon by the sending and receiving applications.

The message structure is based on a complex type. A complex type describes a structure within a message. It contains elements, attributes, and groups that are organized into a hierarchy.

An element can be simple or complex.

- A simple element describes one or more fields in a message. It is based on a simple type. It can repeat, and it can define a default or a fixed value.
- A complex element is a named structure that contains simple elements within the message. Complex elements can contain other complex elements, and they can also contain groups. A complex type defines the content of a complex element.

A simple type describes a class of data within a message. It describes the type of data (for example, string, integer, or float) and it can have value constraints that place limits on the values of any simple elements that are based on that simple type.

A group is a list of elements with information about how those elements can appear in a message. Groups can be ordered (sequence), unordered (all), or selective (choice).



## Message model objects

- **Local** object: Defined and used in only one place
  - Local element within a complex type
  - Local simple type within an element
  - Local complex type within an element
  - Local group within a type
- **Global** object: Increased reusability
  - Referenced from anywhere in the message definition (.mxsd)
  - Referenced from other message definitions in the same message set, by using import or include
  - Has a name that must be unique within message set

© Copyright IBM Corporation 2016

Figure 6-9. Message model objects

WM676/ZM6761.0

### Notes:

Many of the objects in the message model can be either global or local.

A global object must have a unique name, which is used to refer to the object from one or more places in the message model.

Local objects are defined and used in only one place in the message model.

Make objects local unless they must be used in more than one place. A local object reduces the probability of name clashes among the global objects in the message model, and you find it easier to work with the message set.

## Message model object properties

- Logical: Format-independent description of the data
- Physical
  - Controls parsing and writing of the object
  - One set of physical properties for each physical format in the message set (XML, custom wire format, and Tagged/Delimited String formats)
- Documentation: Does not affect processing

© Copyright IBM Corporation 2016

Figure 6-10. Message model objects properties

WM676/ZM6761.0

### Notes:

The logical properties of an object relate to the format-independent description of the concept that is known as the *logical model*. Logical properties describe what data the object contains without saying anything about how it is written down.

The physical properties of an object describe how the object is written down. These properties control the parsing and writing of the object. Each physical format in your message set has one set of physical properties.

## Message set: Container and unit of administration

- Can base one message set on another
- Specify message domains that the message set supports to determine what is generated for deployment to an integration node
- **Message set ID** is automatically created and universally unique

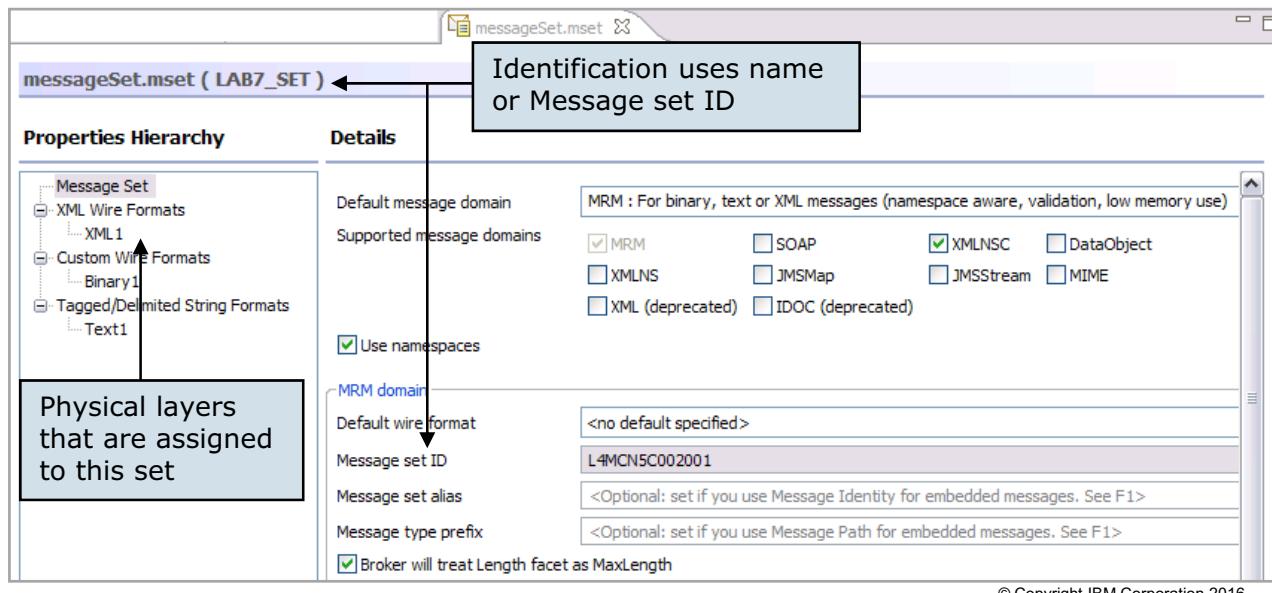


Figure 6-11. Message set: Container and unit of administration

WM676/ZM6761.0

### Notes:

A message set has a name, provided by the folder, and a unique 13-character string that identifies the message set. The name or the identifier can be used interchangeably to specify the message set, such as in an MQRFH2 message header. While the name is more readable, the identifier is unique. The identifier is used in places where you do not have control over the names of the message sets that you are using. After you create message set, the name or the identifier cannot be changed.

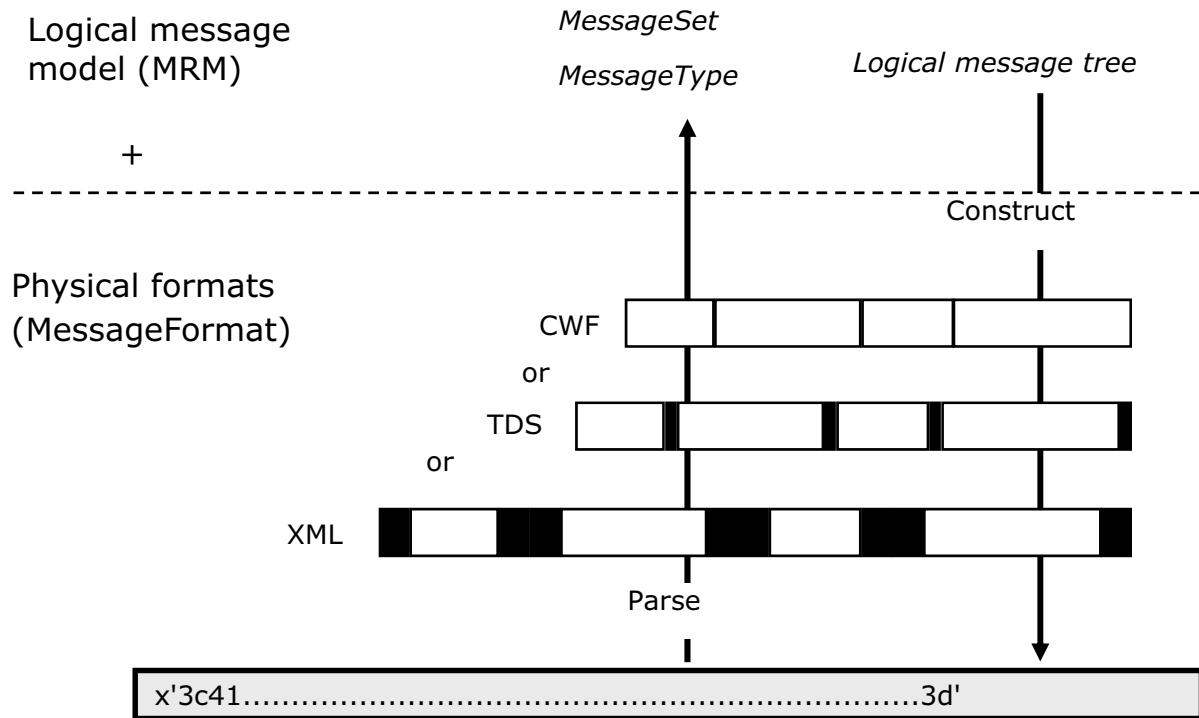
You must specify the message domains that the message set supports. The supported domains determine what is generated for deployment to a broker, and are used when parsing and writing the messages that are defined within the message set.

Multiple message domains can be associated with a single message set. This feature is useful for messages in the MIME domain, where another domain parses the embedded part of the message. It is also useful in the SOAP domain, where the SOAP body will be parsed in the XMLNSC domain after the headers are parsed in the SOAP domain.

By default, message definitions that you create within the message set are XML-namespace-aware.

You are supplied with a collection of predefined, read-only message sets that model the headers that IBM MQ messages use. The predefined message sets are useful for referencing fields in the headers in message flows.

## Physical representation



Key: Filled boxes represent delimiters and tags

© Copyright IBM Corporation 2016

Figure 6-12. Physical representation

WM676/ZM6761.0

### Notes:

The MRM domain uses a language-independent and platform-independent model to reference message structures (that is, the logical message). In this regard, three physical wire format layers are used in the MRM to cover the most popular wire formats:

- Custom wire format (CWF), typically for messages that are composed of fixed-length fields
- Tagged/Delimited String (TDS) wire format
- XML

The physical layer is added to the message set and has a name, for example, CWF1. The broker uses the physical layer on input to aid in parser identification, and on output to convert a message tree into an outgoing bitstream.

At run time, at least one physical layer must be identified in a logical model, but multiple layers are allowed for the same model.

## Custom wire format (binary or CWF)

- Typical data format
  - Example: COBOL or C definitions
- Various physical representations
  - Examples: 4-byte integer, or packed
- Often fixed-length fields, length-encoded or null-delimited
- Variable length and repeating fields
  - Predefined length or repeat
  - Preceding INTEGER field determines dynamic length or repeat counter
- Fixed field order

© Copyright IBM Corporation 2016

Figure 6-13. Custom wire format (binary or CWF)

WM676/ZM6761.0

### Notes:

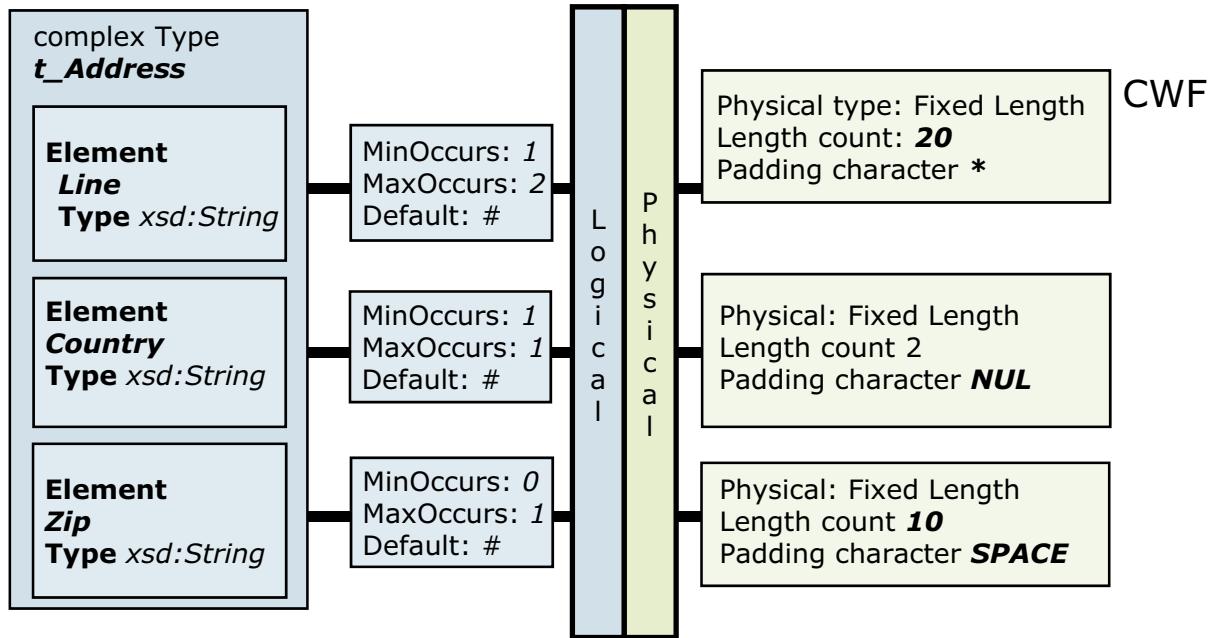
Within a binary (CWF) messaging environment, it is not possible to distinguish one element from the next without knowledge of the message structure. To correctly determine the values of individual elements, the following information must be made available to the message parser:

- The order (defined in the logical properties)
- The length (specified in bytes, characters, or character units)
- The cardinality (the number of repeats)
- The type of data that is contained in each element (partly defined in the logical properties, but can be further qualified in the CWF layer)
- A number of characteristics that are based on the logical type of the data

While the other physical representations support self-defining elements (that is, elements that do not have a definition in the logical model) within the MRM domain, the parsing of a CWF message does not. Any such self-defining elements are discarded during the output of messages in CWF format.

## Logical message and CWF format

```
#*****#*****#
Mail Point 135*****Hursley Park*****UKSO21 2JN
```



© Copyright IBM Corporation 2016

Figure 6-14. Logical message and CWF format

WM676/ZM6761.0

### Notes:

After you define the element and type components, the next step is to specify the connection between complex type and its child elements, referred to as cardinality.

Cardinality identifies the number of occurrences of an element: If an element is mandatory (`MinOccurs=1`), optional (`MinOccurs=0`), or repeating (`MaxOccurs>1`). Connection properties belong to the logical message model and are common to all physical layers. The CWF layer does require all fields to be mandatory and in fixed order on input; it is impossible to parse without tags of some kind.

You must then define the physical CWF properties of the instantiated element within a complex type. CWF properties define the bitstream representation of an instantiated element. A global element might have different CWF properties (a different bitstream) when used within the context of another complex type.

The box at the top of the figure shows the message output. You see that defaults (#) were applied to the HomeAddress structure.

## Tagged/Delimited String Format (text or TDS)

- Typical industry standard formats include SWIFT, EDI, ACORD, AL3
- Typically text, but can also handle binary data
- Variable length, repeat, and field order
- Structured
- Various data separation techniques
  - Fixed length, like CWF
  - Delimited: all elements, or variable length elements only  
Example: CSV (comma-separated values)
  - Tagged

Delimited	Fixed-length	Encoded length
TAG1 : aaa / TAG2 : xx /	TAG1aaaTAG2xxx	TAG103aaaTAG202xx
— Data patterns: Regular expressions, such as [A-Z] {1,3}		

© Copyright IBM Corporation 2016

Figure 6-15. Tagged/Delimited String Format (text or TDS)

WM676/ZM6761.0

### Notes:

The Tagged/Delimited String physical format is designed to model messages that consist of text strings, but it can also handle binary data. Tagged/Delimited String allows a high degree of flexibility when defining message formats, and it is not restricted to model-specific industry standards. So, you can use the Tagged/Delimited String Format to model your own messages.

The fields in the message can have a tag or a label that precedes the data value. The tag is a string that uniquely identifies the data value. With the Tagged/Delimited String Format, you can associate a tag with each element when you define the element.

The message can contain various special characters or strings in addition to the tags and text string data values. The Tagged/Delimited String Format supports a number of different types of special characters or strings. Some messages have a special character or string that separates each data value from the next. In the Tagged/Delimited String Format, this character or string is known as a delimiter.

In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the Tagged/Delimited String Format, this character or string is known as a tag data separator.

A message can be split into a number of substructures in a similar manner to a COBOL or C structure. You can model each of these substructures separately by defining groups, complex types, or elements for each one. A substructure can have a special character or string that indicates its start within the data. This character or string is known in the Tagged/Delimited String Format as a group indicator.

A substructure can also have a special character or string that indicates its end in the data. In the Tagged/Delimited String Format, this character or string is known as a group terminator. A group indicator and group terminator can also be defined for the whole message. Group indicators and group terminators are optional for the message and each substructure.

Some fields within a message can be of fixed length, so a delimiter between each data value is not necessary. The Tagged/Delimited String Format supports this feature.

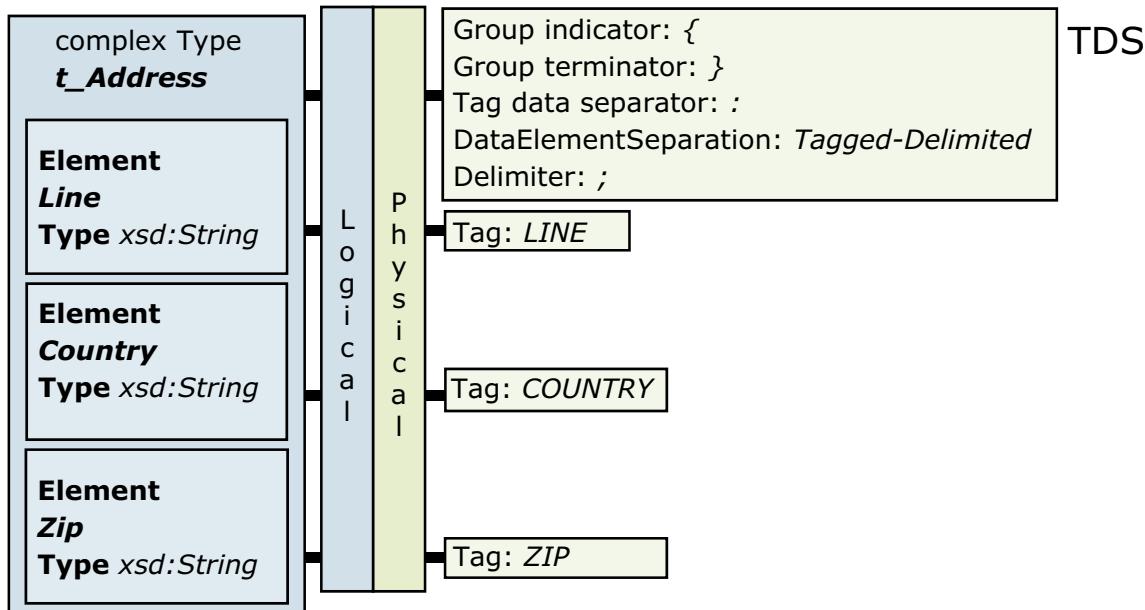
Some tags can be defined as fixed length, so a tag data separator is not necessary.

Data Element Separation is the Tagged/Delimited String property that controls the way elements are separated. It has several options that provide you with the capability to choose; for example, whether tags are used, whether field lengths are fixed or variable, and what types of fields are allowed. The substructures within a message can use different types of data element separation and use different special characters. Therefore, with the Tagged/Delimited String Format you can define different types of data element separation and special characters for each complex type within the message.

If you choose the **Use data pattern** option for Data Element Separation, you can use regular expressions to identify parts of the message data that is assigned to subfields. This option is configured by setting the regular expression in the Data Pattern property.

## Logical message and TDS format

```
{LINE:Mail Point 135;LINE:Hursley Park;COUNTRY:UK;ZIP:SO21 2JN}
```



© Copyright IBM Corporation 2016

Figure 6-16. Logical message and TDS format

WM676/ZM6761.0

### Notes:

The figure shows the physical properties of the AddressesMsg and the resulting output message. The logical message is the same as in the previous example. However, it now has another physical layer for Tagged/Delimited Strings that needs more definition.

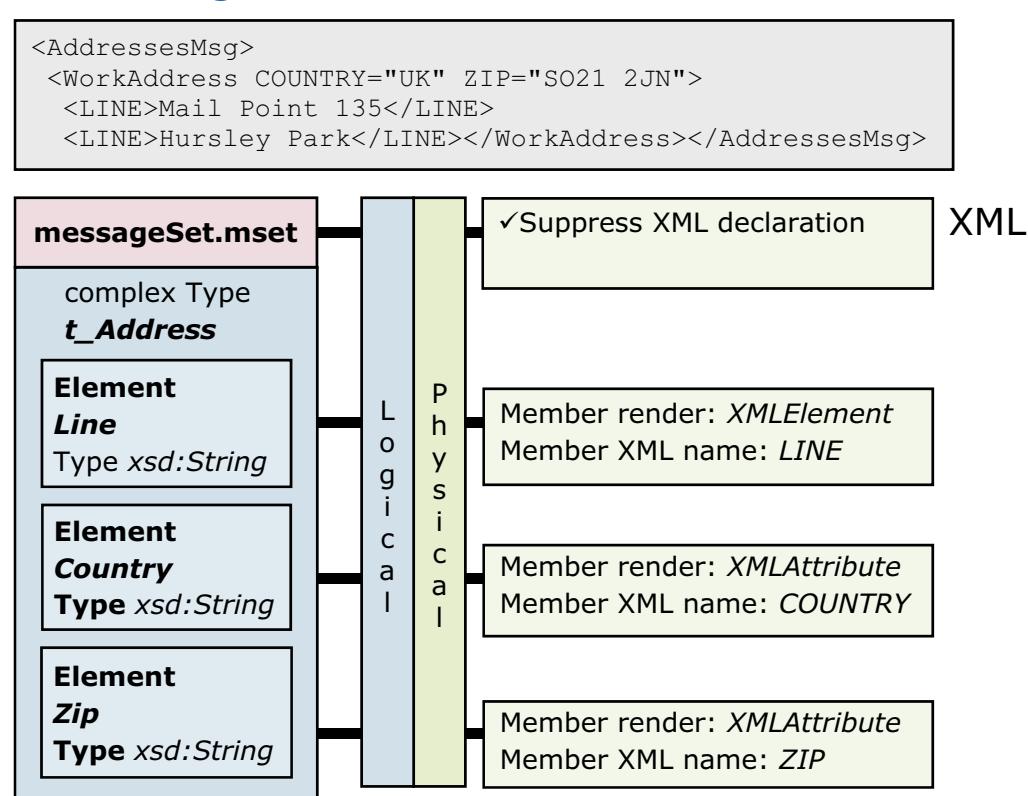
The Tagged/Delimited String properties of the complex type (ADDRESS) define how the data elements of this type are separated.

For each instantiated element within the complex type, you must specify how to find it, which means its tag value. Element tags are only needed for a data element separation of tagged. If you had variable delimited elements, for example, the tag property would not be editable.

A message set can have multiple physical layers of a certain type.

Default values were not applied for the HomeAddress structure because a tagged-delimited structure is valid and can be parsed when elements are missing.

## Logical message and XML format



© Copyright IBM Corporation 2016

Figure 6-17. Logical message and XML format

WM676/ZM6761.0

### Notes:

The figure shows the AddressMsg logical message in XML representation. If messages are in XML, you can use either the XML wire format in the MRM domain, or one of the generic XML domains.

Using MRM-XML gives you extra control over the rendering of your data. For example, you might have a data field that is rendered as an XML element in one message, and as an XML attribute in another message. Or you can have a data field that is known by a particular name in one message, and a different name in another message. This advanced rendering can be specified by using XML wire format properties.

MRM-XML provides you with the capability to share a common logical message structure between physical formats. For example, if you have a message that a COBOL application creates, you can use an XML wire format to easily and quickly define the equivalent XML message. You can also create your message model by importing your COBOL copybook.

You can generate a message dictionary for use by the MRM parser in IBM Integration Bus. The message dictionary enables the MRM parser to interpret the data in an XML message in an advanced manner. For example, the dictionary indicates the real data type of a field in the message, instead of always treating it as a character string. It also enables the MRM parser to

validate XML messages against the dictionary. This technique provides similar capabilities as XML validation against an XML DTD or schema.

XML messages are, by their nature, self-describing: a tag name or an attribute name prefixes each piece of data. Therefore, it is possible for an XML message instance to contain elements that are not in the MRM definition for that message. If such an element exists in the message set, the MRM objects for that element are used in parsing or writing the message. If the element does not exist in the message set, it is treated as a self-defining element and its data type is set to STRING.

## Default and fixed values for elements

- Logical property of element
- Integration Bus applies default or fixed values when necessary to create parseable bitstream
  - For CWF and TDS fixed-length output messages
  - Can lead to unexpected validation errors for missing elements in other formats, even though they have default or fixed values
- ESQL can be used to provide default values
  - COALESCE saves coding IF-THEN-ELSE constructs
  - Returns first non-NULL value of argument list

```
SET OutputRoot.MRM.F1=COALESCE(InputBody.F1,0);
```
- Can temporarily convert to CWF to get default values applied
  - Use ResetContentDescriptor node to reparse by using the message domain, set, type, and format that are supplied in node properties

© Copyright IBM Corporation 2016

Figure 6-18. Default and fixed values for elements

WM676/ZM6761.0

### Notes:

Earlier in the course, you saw that you can specify default values when you define logical and physical models. Those default values are applied only under certain conditions. For example, if you defined a field to have a default value in a message set, but during processing that field is not parsed, the default value is not applied.

You can also provide default values programmatically in a compute-type node, or by using the ResetContentDescriptor node, which forces parsing of the message with a different parser.

Using the ResetContentDescriptor node can be useful if you want to dynamically reparse a message at run time. For example, some generic error handler routines reparse a message into BLOB format so that it can be saved (to a file, database, or queue) without regard to the data types in the message body.

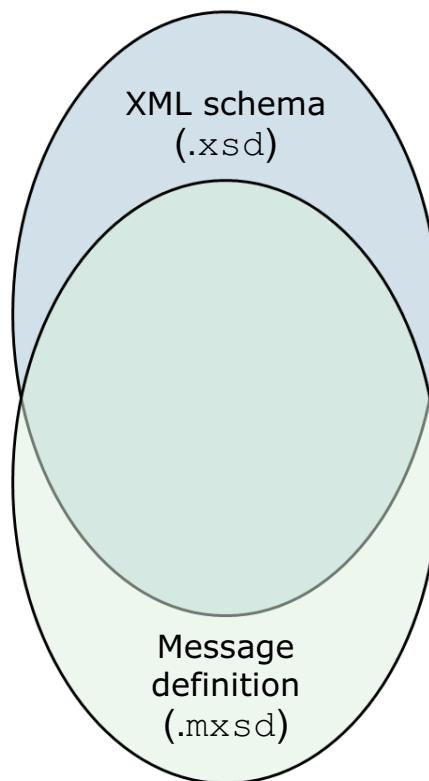
## MRM message model versus XML schema

### XSD properties

- Redefinition not supported (validation error)
- Without effect
  - White space facets
  - Pattern facets on simple type other than xsd:string
  - Identity constraints (unique, key, key reference)

### Message definition extensions to XSD

- Messages
- Extra compositions
  - OrderedSet
  - UnorderedSet
  - Message
- Physical format information



© Copyright IBM Corporation 2016

Figure 6-19. MRM message model versus XML schema

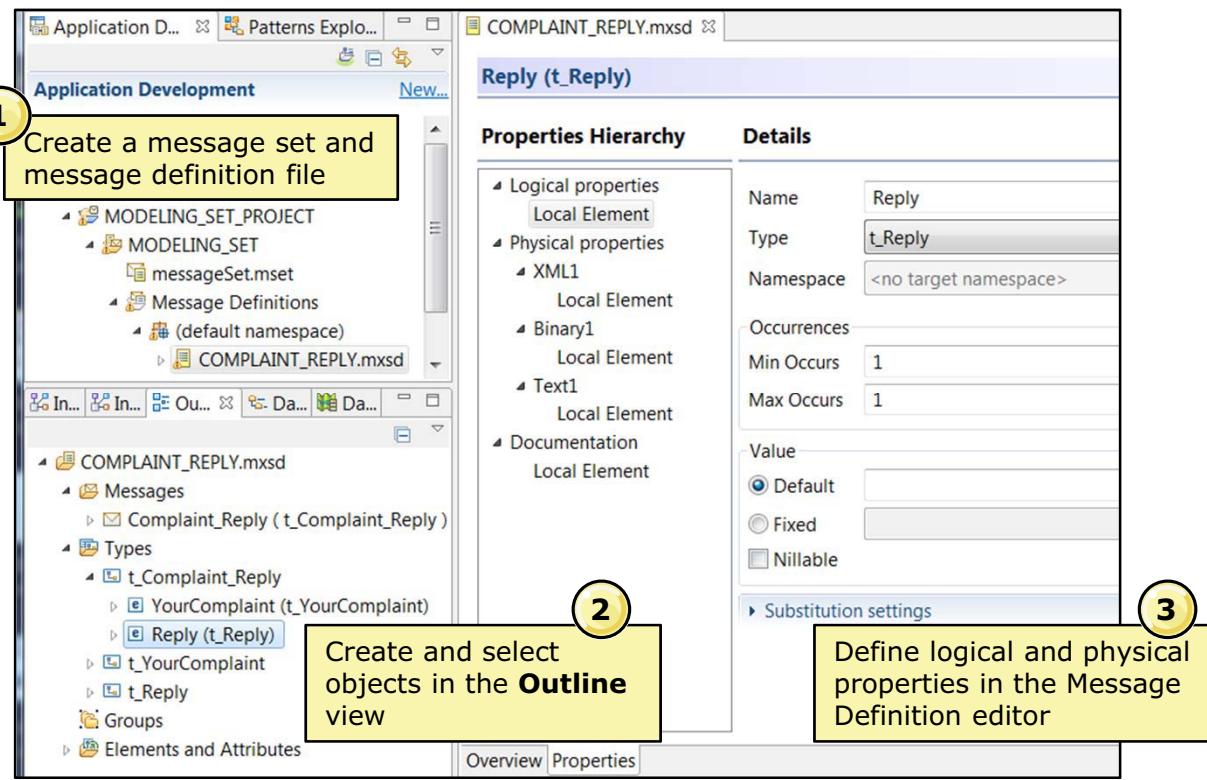
WM676/ZM6761.0

### Notes:

The Integration Bus logical message model is an XML schema, with a few extensions and restrictions.

XML schema constructs **key**, **keyref**, and **unique** enforce uniqueness constraints on the value of an element or attribute in an instance document within a certain scope. They use XPath expressions to select a set of elements and attributes, which are the target of uniqueness constraints. They do not participate in building the structural contents of the message. These constructs can be imported without change.

## Create and populate message definition



© Copyright IBM Corporation 2016

Figure 6-20. Create and populate message definition

WM676/ZM6761.0

### Notes:

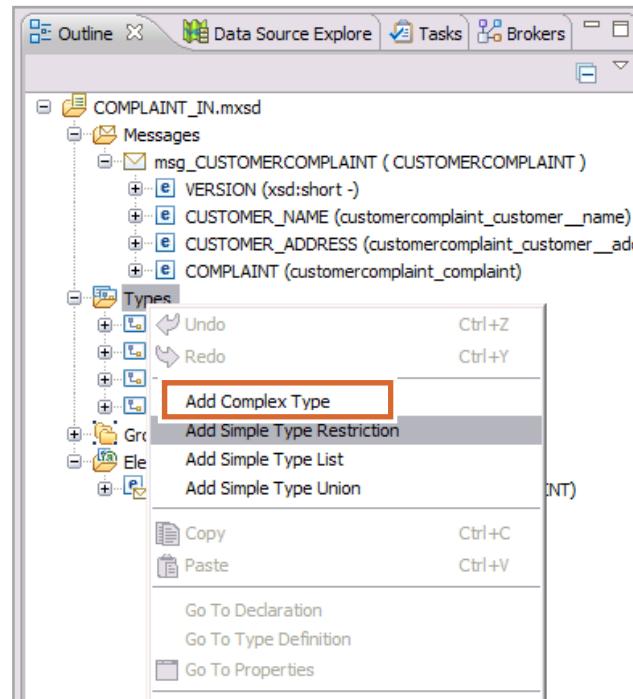
Objects inside the Message Definition file are not visible in the **Application Development** view. Instead, you work in the **Outline** view or the Message Definition editor.

The figure shows the main steps for creating and populating the message definition.



## Bottom-up approach for message definition

1. Start with lowest level complex types by selecting **Add Complex Type** in the **Types** folder.
2. Add local elements to these complex types, or create global elements by selecting **Add Global Element** in the **Elements and Attributes** folder.
3. Add element reference to selected type.
4. Add message and change type from `complexType1`.



© Copyright IBM Corporation 2016

Figure 6-21. Bottom-up approach for message definition

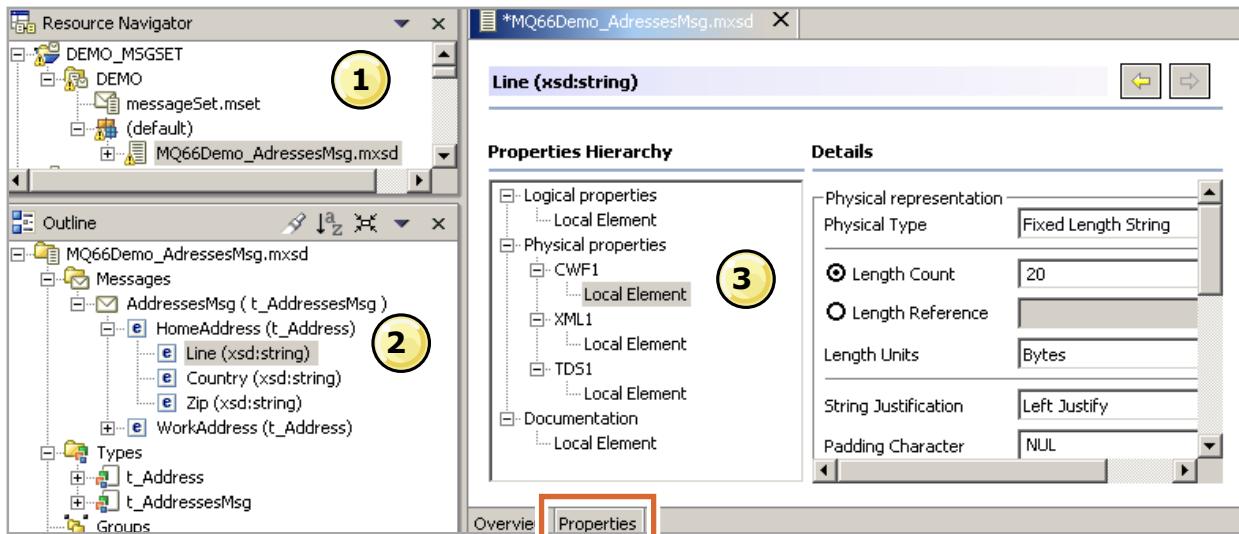
WM676/ZM6761.0

### Notes:

You can define a message in many ways, such as using an importer or through manual creation. This figure shows the major steps for creating a message definition by starting with the lowest complex type and working up to the highest complex type.



## Setting physical properties



1. Select the `*.mxsd` file in the **Application Development** view.
2. In the **Outline** view, select the object to customize its element properties.
3. In the editor view, define physical properties for one or more physical formats.

© Copyright IBM Corporation 2016

Figure 6-22. Setting physical properties

WM676/ZM6761.0

### Notes:

This figure shows the steps for setting the physical properties of a CWF format message on the **Properties** tab:

1. Select the message set definition file (`.mxsd`) in the **Integration Toolkit Application Development** view.
2. In the **Outline** view, select the object in the message set definition file to customize its properties.
3. In the editor view, select **CWF1 > Local Element** from the **Properties** tab.

WebSphere Education

## Value constraints

The screenshot shows the 'Properties Hierarchy' pane on the left, listing various schema components like CustomerName, Address, Line, Town, Country, Zip, Complaint, and xsd:string. Under 'Country', 'xsd:string' is selected. The 'Details' pane on the right contains sections for 'Length Constraints' (Length set to 2), 'Min' (empty), 'Max' (empty), 'White Space' (dropdown menu), 'Enumerations' (list of US, DE, FR, UK with 'Add' and 'Delete' buttons), and 'Patterns' (list of '[A-Z]2' with an 'Add' button). A callout box highlights the 'Value Constraints' node under Logical properties.

**Property of custom (restricted) simple type**

- Length constraints: Length, minimum, maximum
- Inclusive and exclusive constraints: Minimum, maximum
- White space: Preserve, replace, collapse
- Enumerations
- Fraction digits, total digits
- Patterns: Regular expressions like [^abc] or a{2,4}

© Copyright IBM Corporation 2016

Figure 6-23. Value constraints

WM676/ZM6761.0

### Notes:

Value constraints are normally associated with a custom simple type; they refine a simple type by defining limits on the values that it can represent.

The properties that are shown on the object page and the values that those properties can take can vary according to the type of the object. For example, the properties **Fraction Digits** and **Total Digits** are available only for decimal simple types.

An enumeration restricts which values can be set for the value constraint, for example, "ABC" and "123."

A pattern defines a string that is used as a regular expression that the data in the associated type must match. The regular expression syntax that is supported is a subset of XML schema regular expressions.

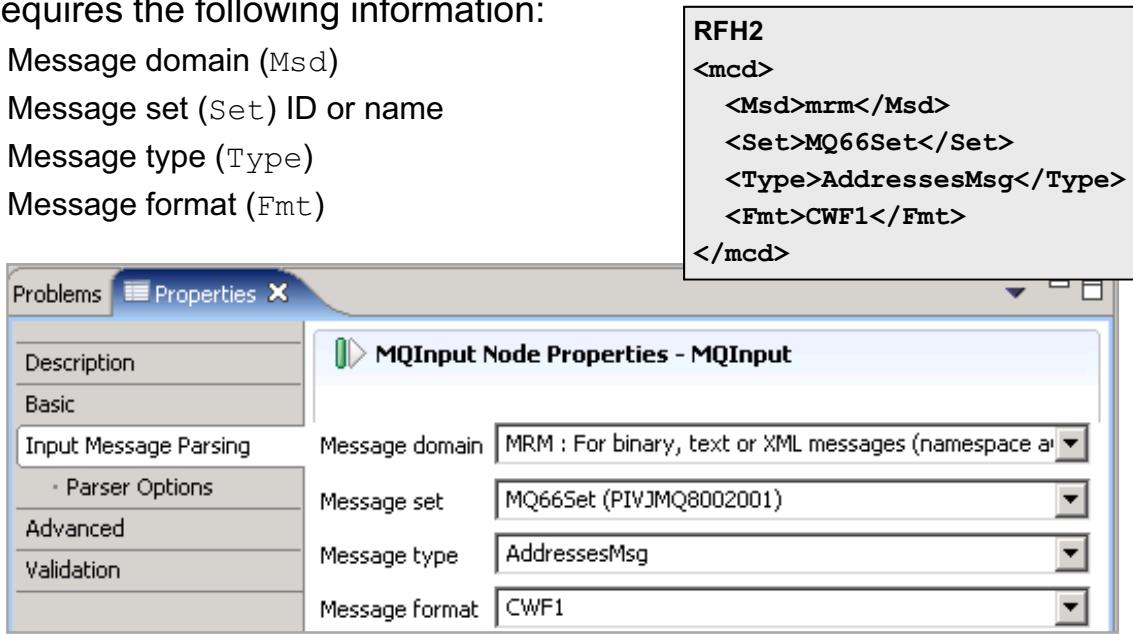
To configure value constraints, click the simple type that you are updating in the Outline view and click the **Properties** tab. In the Properties hierarchy under **Logical Properties**, click **Value Constraints**. The current value constraints settings for the selected simple type are shown in the **Details** pane.

## 6.2. Referencing the MRM message in a message flow

WebSphere Education 

## Assigning the parser to a message (1 of 2)

- Integration Bus uses `Root.Properties` to determine parser and dictionary for parsing (In) and serialization (Out)
- Requires the following information:
  - Message domain (`Msd`)
  - Message set (`Set`) ID or name
  - Message type (`Type`)
  - Message format (`Fmt`)



© Copyright IBM Corporation 2016

Figure 6-24. Assigning the parser to a message (1 of 2)

WM676/ZM6761.0

### Notes:

Deploying a message set to the integration node is the same as for message flows; add the message set to a BAR file. If you are using an Integration Bus application, deploying the application automatically deploys the message set. If you attempt to deploy the message set by itself, the toolkit suggests that the entire application must be deployed.

For each incoming message, the integration node must be instructed on how to parse it. To do so, you assign a parser to the message. You generally assign the parser at design time in the input node properties. A parser can also be assigned dynamically at run time.

## Assigning a parser to a message (2 of 2)

- On input, integration node provides parser properties based on the following hierarchy:
  1. MQRFH2 Header (`mcd` folder) to identify the type of message so it can be “assigned” to the correct parser
  2. MQMD.Format
  3. Default values in the MQInput node
  4. BLOB
- On output, Integration Bus uses properties that the transformation nodes set in the message flow

© Copyright IBM Corporation 2016

Figure 6-25. Assigning a parser to a message (2 of 2)

WM676/ZM6761.0

### Notes:

The MQRFH2 header in the message identifies the incoming message so it can be assigned to the correct parser. The header contains a series of name-value parameters. These attributes include **Domain**, **Message Set**, **Message Type**, and **Message Format**. You can specify some or all of the values in the MQRFH2.

If your applications do not supply an RFH2 header and you want to use the MRM parser, you must set the defaults in the input node.

If no parser information is available, then the default is to treat the message as a BLOB, so the message body is not parsed at all.



## MRM parser operation

- Depends on the physical format that is associated with the input or output message:
  - For a binary message, the parser reads a set sequence of bytes according to information in the CWF physical format
  - For a text message, the parser uses a separator to parse each portion of the message bitstream
  - For an XML message, the parser reads the XML markup language (element tags and attributes), guided by information in the XML physical format

© Copyright IBM Corporation 2016

---

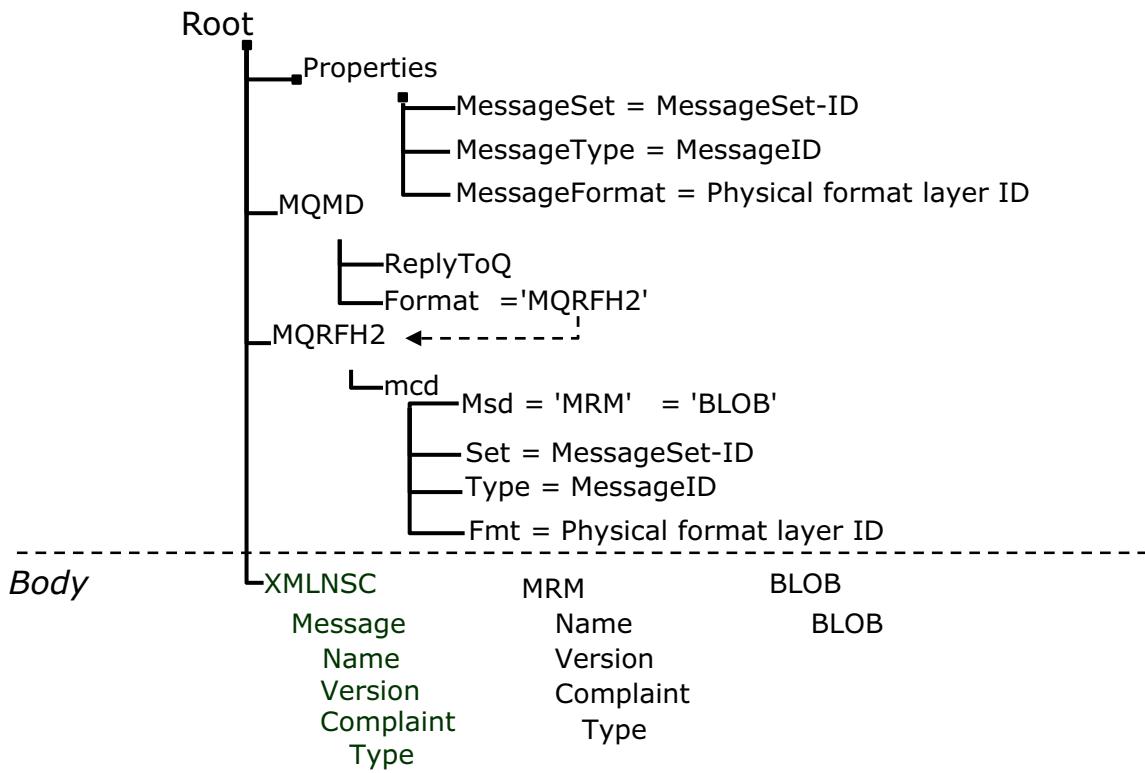
Figure 6-26. MRM parser operation

WM676/ZM6761.0

### Notes:

The MRM parser operation depends on the physical format and model properties for the data.

## Logical message tree: Root



© Copyright IBM Corporation 2016

Figure 6-27. Logical message tree: Root

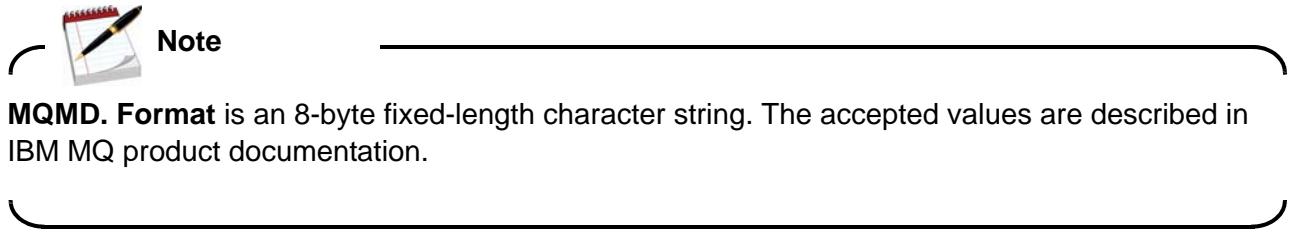
WM676/ZM6761.0

### Notes:

The figure shows the logical message tree that the integration node builds when it parses a message, depending on the message domain.

The first branch, **Properties**, is present only while the message is processed within the integration node. It is not part of the actual IBM MQ message. It contains information for predefined messages so that the parser knows where to get the formats. The domain is contained in the last branch of Root, the section where the application data is located.

The various IBM MQ headers are between **Properties** and **Body**, chained together by the Format field.



## Sample Compute node ESQL for MRM

```

CREATE COMPUTE MODULE Output_in_3_formats
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    CALL CopyMessageHeaders();
    SET OutputRoot.Properties.MessageSet='My_Set';
    SET OutputRoot.Properties.MessageType='AddressesMsg';
    -- omit HomeAddress fields
    SET OutputRoot.MRM.WorkAddress.Line[1]='Mail Point 135';
    SET OutputRoot.MRM.WorkAddress.Line[2]='Hursley Park';
    SET OutputRoot.MRM.WorkAddress.Country='UK';
    SET OutputRoot.MRM.WorkAddress.Zip='SO21 2JN';

    SET OutputRoot.Properties.MessageFormat='Binary1';
    PROPAGATE DELETE NONE;

    SET OutputRoot.Properties.MessageFormat='XML1';
    PROPAGATE DELETE NONE;

    SET OutputRoot.Properties.MessageFormat='Text1';
    RETURN TRUE;
END;

```

ESQL to build **AddressesMsg**  
and output the same content in  
three different physical formats

**PROPAGATE DELETE**  
NONE propagates the  
message but does  
not delete the output  
buffer

**RETURN TRUE** identical  
to PROPAGATE

© Copyright IBM Corporation 2016

Figure 6-28. Sample Compute node ESQL for MRM

WM676/ZM6761.0

### Notes:

The figure shows the ESQL code that generates the AddressesMsg sample output in three different physical formats.

In addition to filling the MRM body and setting Properties for output, this module uses the PROPAGATE statement to create multiple different output messages from a single Compute node.

JavaCompute node has the equivalent  
`com.ibm.broker.plugin.MbOutputTerminal.propagate()` method.



## Unit summary

Having completed this unit, you should be able to:

- Describe the physical message formats that can be defined with an MRM message set
- Configure the logical and physical message properties
- Reference an MRM model in ESQL

© Copyright IBM Corporation 2016

Figure 6-29. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or False: Default properties of the physical formats of a message set can cause warnings in the IBM Integration Bus task list.
  
2. Which three values identify a dictionary for an MRM message?
  - a. Parser
  - b. Message Set
  - c. Message Name
  - d. Format Name

© Copyright IBM Corporation 2016

Figure 6-30. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

1.

2.

## Checkpoint answers

1. True or False: Default properties of the physical formats of a message set can cause warnings in the IBM Integration Bus task list.  
Answer: **True**. For custom-wire and Tagged/Delimited strings, default settings are not complete and must be customized. For example, for all CHAR elements, physical length must be added.
  
2. Which three values identify a dictionary for an MRM message?
  - a. Parser
  - b. Message Set
  - c. Message Name
  - d. Format Name

Answer: **b, c, and d**

© Copyright IBM Corporation 2016

Figure 6-31. Checkpoint answers

WM676/ZM6761.0

### Notes:



# Unit 7. Supporting web services

## What this unit is about

You can use IBM Integration Bus nodes and services to connect to other web services providers and consumers. This unit describes the use of web services transport and WSDL generation from message definitions. It also covers handling messages with SOAP formats.

## What you should be able to do

After completing this unit, you should be able to:

- Access a message flow as a web service
- Request a web service from within a message flow
- Describe the functions of HTTP and SOAP nodes
- Generate WSDL files from message sets
- Describe the SOAP message tree
- Describe how WS-Addressing and WS-Security are implemented in IBM Integration Bus

## How you will check your progress

- Checkpoint
- Exercise

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Access a message flow as a web service
- Request a web service from within a message flow
- Describe the functions of HTTP and SOAP nodes
- Generate WSDL files from message sets
- Describe the SOAP message tree
- Describe how WS-Addressing and WS-Security are implemented in IBM Integration Bus

© Copyright IBM Corporation 2016

---

Figure 7-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- Developing web service and web service clients in Integration Bus
- HTTP nodes
- SOAP nodes
- WS-Addressing and WS-Security

© Copyright IBM Corporation 2016

Figure 7-2. Topics

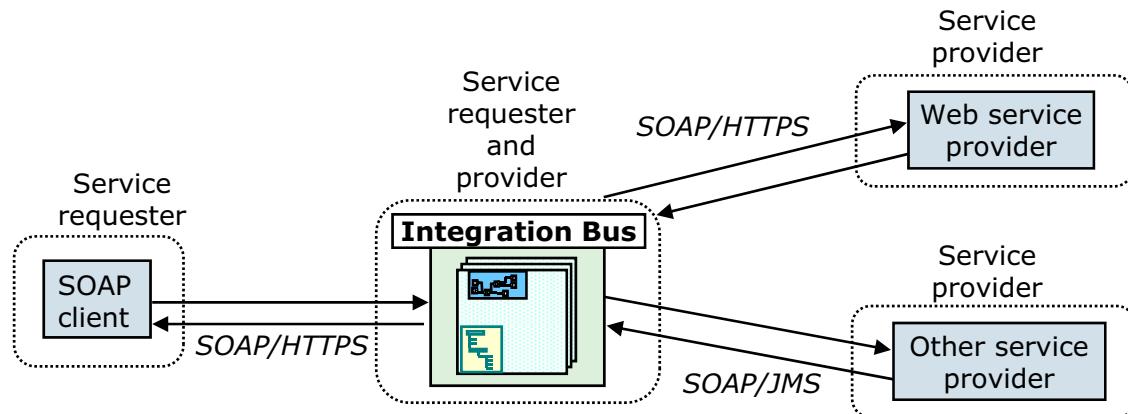
WM676/ZM6761.0

## Notes:



## **7.1. Developing web service and web service clients in Integration Bus**

## IBM Integration Bus and web services



- Integration Bus can act as front end to many service providers to allow for failover and provide higher overall quality of service (QoS)
- Message logging for audit or nonrepudiation
- Map SOAP to other industry standard format
- Selection of service provider-based on message context and content

© Copyright IBM Corporation 2016

Figure 7-3. IBM Integration Bus and web services

WM676/ZM6761.0

### Notes:

IBM Integration Bus is a convenient central point for brokering web services. You can wrap existing services or access web services from existing applications.

Integration Bus can act as a SOAP intermediary by providing first point-of-contact functions, such as hiding and changing service implementation, transformation between WSDL definitions, monitoring, data warehousing, and auditing. So, normal integration node strengths are applied to and enhance web services.

## Approaches to developing web services

- An **integration service** is a specialized application with a defined interface that acts as a container for SOAP web services
  - Developers focus on implementing service operations instead of message and transport protocols
  - Define a service interface with WSDL
- For more fine-grained control over developing a web service, create message flows with SOAP nodes directly
  - Build SOAP web services with a JMS transport
  - Build a gateway for multiple web services
- To support other types of web services, create message flows with HTTP nodes
  - Build REST web services
  - Build XML web services that do not use SOAP messages, such as XML-RPC (Remote Procedure Call) services

© Copyright IBM Corporation 2016

Figure 7-4. Approaches to developing web services

WM676/ZM6761.0

### Notes:

The Integration Toolkit supports three approaches to developing web services.

The integration service is a specialized application that acts as a container for SOAP web services that are defined with a WSDL document. Integration services provide a structured solution for quickly building a web service. The integration service container manages the SOAP message parsing and HTTP message transport options. With this approach, you focus on implementing business logic, not managing the lower-level message serialization and deserialization tasks. Integration services are described in a separate unit.

If you require a finer level of control in processing SOAP web services, you can build message flows with SOAP nodes. With this approach, you can create a SOAP web service that non-HTTP transport protocols, such as JMS.

The third option is to build a web service by using HTTP nodes. For example, RESTful web services and XML-RPC web services do not conform to the SOAP specification. The integration service container and SOAP nodes do not support these web service types.

## Types of web services

- SOAP web services
  - SOAP specification defines an XML-based message format for web services
  - Within the SOAP message envelope, the SOAP header stores message routing, security, transaction, and quality of service information
  - SOAP message body stores the message payload in XML format
- XML-RPC web services
  - An older form of RPC style web service
  - Request message consists of a simple XML message with a procedure name and its parameters
- RESTful web services
  - REST models web services as server resources, as opposed to procedures from a server-based application
  - REST services use the HTTP header and body as its message format
  - JavaScript Object Notation (JSON) is a common data format for REST messages

© Copyright IBM Corporation 2016

Figure 7-5. Types of web services

WM676/ZM6761.0

### Notes:

The three most common types of web services that an Integration Bus developer encounters are SOAP web services, XML-RPC web services, and RESTful web services.

XML-RPC is an older form of a remote procedure style web service. Introduced in 1998, XML-RPC were popular with Microsoft and webMethods web service implementations.

SOAP web services provided a standard XML message format and encoding rules. SOAP supports both remote procedure call and document style web services.

Lastly, REST models web services as server resources, instead of a set of procedures from a server application.

In this unit, you learn which message nodes to use to start and support each web service type.

## When to use SOAP nodes? When to use HTTP nodes?

- Use SOAP nodes and SOAP domain for SOAP-based web services
  - Common SOAP message assembly, regardless of bitstream format of message
  - Built-in support for SOAP headers, WS-Addressing, and WS-Security
  - Automatic processing of SOAP with Attachments (SwA) and Message Transmission Optimization Mechanism (MTOM) messages
  - Runtime checking against WSDL
- Use HTTP nodes and XMLNSC domain when message flow:
  - Uses single request node to handle multiple SOAP requests and responses from more than one WSDL
  - Interacts with web services that use different standards (such as REST or XML-RPC)
  - Does not use SwA, MTOM, WS-Addressing, or WS-Security

© Copyright IBM Corporation 2016

Figure 7-6. When to use SOAP nodes? When to use HTTP nodes?

WM676/ZM6761.0

### Notes:

You can use both SOAP message processing nodes and HTTP message processing nodes when you write message flows that use web services. When is it better to use one over the other?

If the messages you process are based on SOAP, it is better to use SOAP nodes and the SOAP domain. These provide more capability for handing SOAP messages than do HTTP nodes. SOAP nodes use a common SOAP message assembly, regardless of the content of the message. SOAP nodes automatically handle SOAP with Attachments (SwA) and Message Transmission Optimization Mechanism (MTOM) messages, two common SOAP message formats. SOAP nodes can also check the messages against the WSDL that was used to create the message flow components at run time.

If the messages are not based on SOAP, or if a message flow handles messages that use more than one WSDL, you can use HTTP nodes and the XMLNSC domain. This alternative is also best if your message flow interacts with multiple web services that use different standards.

## HTTP listeners

- You can choose between integration node listeners and integration server (embedded) listeners to manage HTTP messages in your HTTP or SOAP flows
- Integration node listener
  - Requires access to SYSTEM.BROKER queues on an IBM MQ queue manager that is specified on the integration node
  - Default ports: HTTP connector = 7080, HTTPS connector = 7083
- Integration server embedded listener
  - Integration server embedded listener does not require IBM MQ
  - Default port range: HTTP connector = 7800 – 7842, HTTPS connector is 7843 – 7884
- Choice of listener affects message flows that handle inbound web service requests by using SOAP and HTTP nodes
  - By default, SOAP Input, SOAP Reply, and SOAP AsyncResponse nodes use the integration server listener
  - By default, HTTP nodes use the integration node listener
  - If you disable the integration node listener, the integration server listeners are used for all HTTP and SOAP nodes, even if you did not explicitly enable them

© Copyright IBM Corporation 2016

Figure 7-7. HTTP listeners

WM676/ZM6761.0

### Notes:

You can choose between integration node listeners and integration server (embedded) listeners to manage HTTP messages in your HTTP or SOAP flows.

The integration node listener requires access to system queues on the queue manager that is specified on the integration node. Therefore, if you want to use an integration node listener, you must install IBM MQ Server. If you use HTTP nodes or SOAP nodes with the integration server embedded listener, IBM MQ is not required.

By default, SOAP Input, SOAP Reply, and SOAP AsyncResponse nodes use the integration server listener.

## 7.2. HTTP nodes

## HTTP message processing nodes

- To develop a web service in a message flow:
  - HTTP Input receives an HTTP request message from the client
  - HTTP Reply returns an HTTP response message to the client
- To develop a web service client in a message flow:
  - HTTP Request sends an HTTP request message to an HTTP client
- If you use HTTP nodes to process SOAP web services, use SOAP Extract and SOAP Envelope message processing nodes to create and modify a SOAP message

© Copyright IBM Corporation 2016

Figure 7-8. HTTP message processing nodes

WM676/ZM6761.0

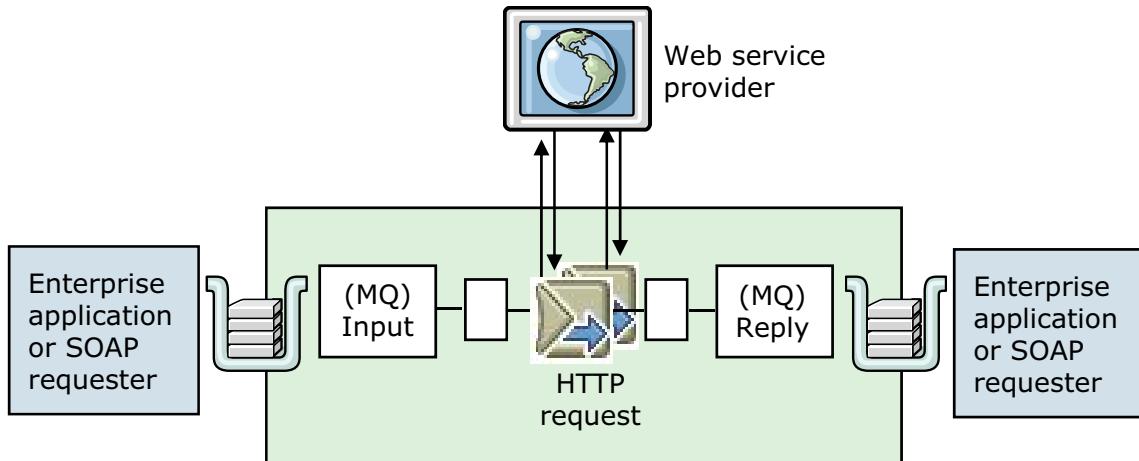
### Notes:

HTTP nodes support web service provider and consumer scenarios with request/response.

If you use HTTP nodes and want to work only on the payload in the SOAP body, you would use the SOAPExtract and SOAPEnvelope nodes.

## Message flow as HTTP/HTTPS client

- Give web service access to established clients
- Integration Bus acts as a buffer between changing set of service providers (web service and others)
- Aggregation of web services



© Copyright IBM Corporation 2016

Figure 7-9. Message flow as HTTP/HTTPS client

WM676/ZM6761.0

### Notes:

Use the **HTTPRequest** node to interact with a web service by using all or part of the input message as the request sent to that service. You can also configure the node to create an output message from the contents of the input message. You can then augment this message with the contents of the web service response before you propagate the message to subsequent nodes in the message flow.

Depending on the configuration, this node constructs an HTTP or an HTTP over SSL (HTTPS) request from the specified contents of the input message. It sends this request to the web service. It receives the response from the web service and parses the response for inclusion in the output tree. If the configuration requires HTTP headers, it generates them.

The **HTTPRequest** node does full HTTP request/reply synchronously. It has options to build the request header automatically. If it is automatically built, it includes an empty **SOAPAction**. Content-length is provided automatically on request. It handles only HTTP V1.0 requests.

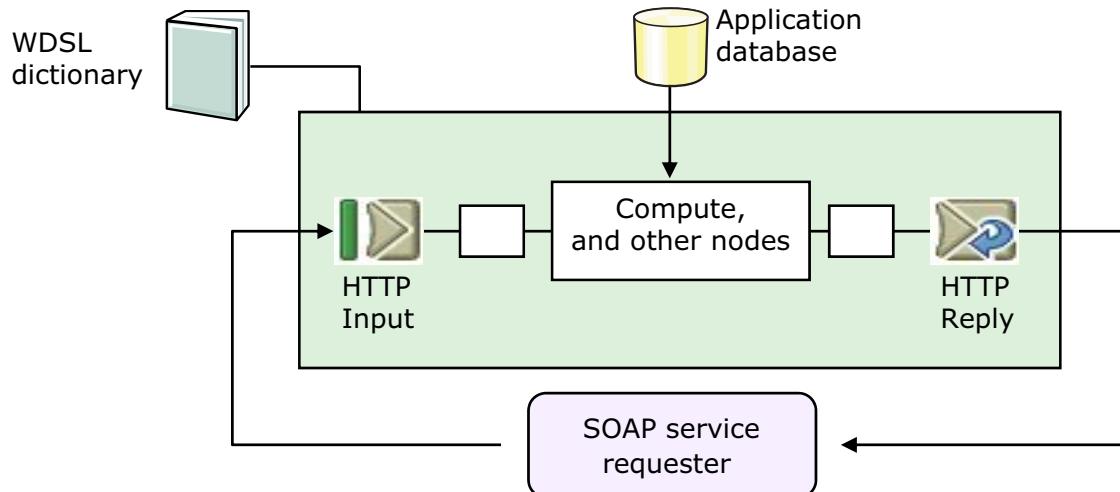
You can specify the format of output messages in the node. It currently generates an exception for a 4XX HTTP response code. Exceptions are generated on TCP/IP errors.

You can set the property **Default Web service URL** on the `HTTPRequest` node to determine the destination URL for a web service request.

You can configure a `Compute` node before the `HTTPRequest` node within the message flow to override the value that is set in the property. In the `Compute` node, code ESQL that stores a URL string in `LocalEnvironment.Destination.HTTP.RequestURL`. The `HTTPRequest` node retrieves the URL string and uses it in place of the node property value. You can also set the request URL in the special header `X-Original-HTTP-URL` in the `HTTPRequestHeader` section of the request message (which overrides all other settings) in a `Compute` node. However, you might want to use the `LocalEnvironment` content for this purpose.

## Message flow as service provider

- Integration Bus acts as intermediary to the web service to provide routing
- Many more combinations



© Copyright IBM Corporation 2016

Figure 7-10. Message flow as service provider

WM676/ZM6761.0

### Notes:

Each integration node has a single TCP/IP port on which incoming HTTP requests are accepted. Client applications can post to a predefined URL or a URL that was already looked up. An attribute on each HTTPInput node is used to qualify the requests for which the node is responsible.

A message flow can act as an intermediary and transform, route, or aggregate web service requests. A message flow parses the request, and forwards it on to one or more destinations that are based on content, perhaps after transformation. Reply data from the destinations is returned to the original client, possibly after more transformation or aggregation.

A message flow can transform HTTP to IBM MQ. It parses the incoming HTTP request, and forwards the transformed request onto an IBM MQ application that does not handle HTTP. The reply is then transformed into the native client format and returned.

## HTTP nodes main properties

- HTTP Input node
  - Web service URL
  - Use HTTPS
  - Maximum client wait time for HTTP Reply, on timeout send SOAP fault message
  - Security properties
- HTTP Reply node
  - Reply sends timeout for acknowledgment from client
  - Generates default HTTP headers from HTTP ReplyHeader or HTTP ResponseHeader of input message
- HTTP Request node
  - Web service URL
  - Request timeout for web services replies, default = 120 seconds
  - HTTP proxy settings
  - Use `InputBody` as request, or path (for example, `InputRoot.x`)
  - Use web service reply as `OutputBody`, or path (for example, `OutputRoot.x`)
  - Generate default HTTP headers from Input

© Copyright IBM Corporation 2016

Figure 7-11. HTTP nodes main properties

WM676/ZM6761.0

### Notes:

When a message is received from a web service client or a web server, the `HTTPInput` or `HTTPRequest` node that receives the message must parse the HTTP headers to create elements in the message tree. When an `HTTPReply` or `HTTPRequest` message sends a message to a web service client or web server, it parses the HTTP headers from the message tree into a bitstream.

The `HTTPInput`/`HTTPRequest` node normally creates the `HTTPInput` and `HTTPResponse` headers from the original client/server bitstream.

You can either create `HTTPReply` and `HTTPRequest` headers by using a `Compute` node, or the respective nodes (`HTTPReply` or `HTTPRequest`) add a default header set.

Remember to add appropriate MQMD headers when you send a message from an `HTTPInput` node to the `MQOutput` node.

## 7.3. SOAP nodes

## SOAP message processing nodes

- When message flow is web services provider:
  - SOAP Input
  - SOAP Reply
- When message flow is web services consumer:
  - SOAP Request
  - SOAP AsyncRequest and SOAP AsyncResponse
- When message flow is either SOAP extract or SOAP envelope
- WSDL plays major role in defining and configuring the flow
  - Drag to create skeleton provider and consumer flows
  - Configure explicitly on node
- Advanced capabilities are configured on nodes
  - WS-Addressing
  - WS-Security

© Copyright IBM Corporation 2016

Figure 7-12. SOAP message processing nodes

WM676/ZM6761.0

### Notes:

SOAP nodes process SOAP messages directly. For the consumer scenario, both synchronous and asynchronous capability is supported. This capability allows the consumer within the flow to either block, wait for the response, or alternatively continue down the flow.

SOAP nodes support web service provider and consumer scenarios with request/response.

## SOAP Input and SOAP Reply nodes

- Analogous to HTTP nodes, except:
  - Configured by WSDL
  - Supports WS-Addressing and WS-Security
  - Supports SOAP V1.1/V1.2, WSDL V1.1, MTOM/XOP, SOAP with Attachments
- All data is organized in a SOAP domain in the message tree
- SOAP Reply node sends result back to requester
  - Extracted SOAP headers are automatically restored if appropriate
  - SOAP Envelope node can insert headers before this node if they do not exist
- Every integration server that contains a SOAP Input node is allocated a TCP/IP port; allows incoming HTTP requests to be accepted
  - Default port range is configured at the integration node level

© Copyright IBM Corporation 2016

Figure 7-13. SOAP Input and SOAP Reply nodes

WM676/ZM6761.0

### Notes:

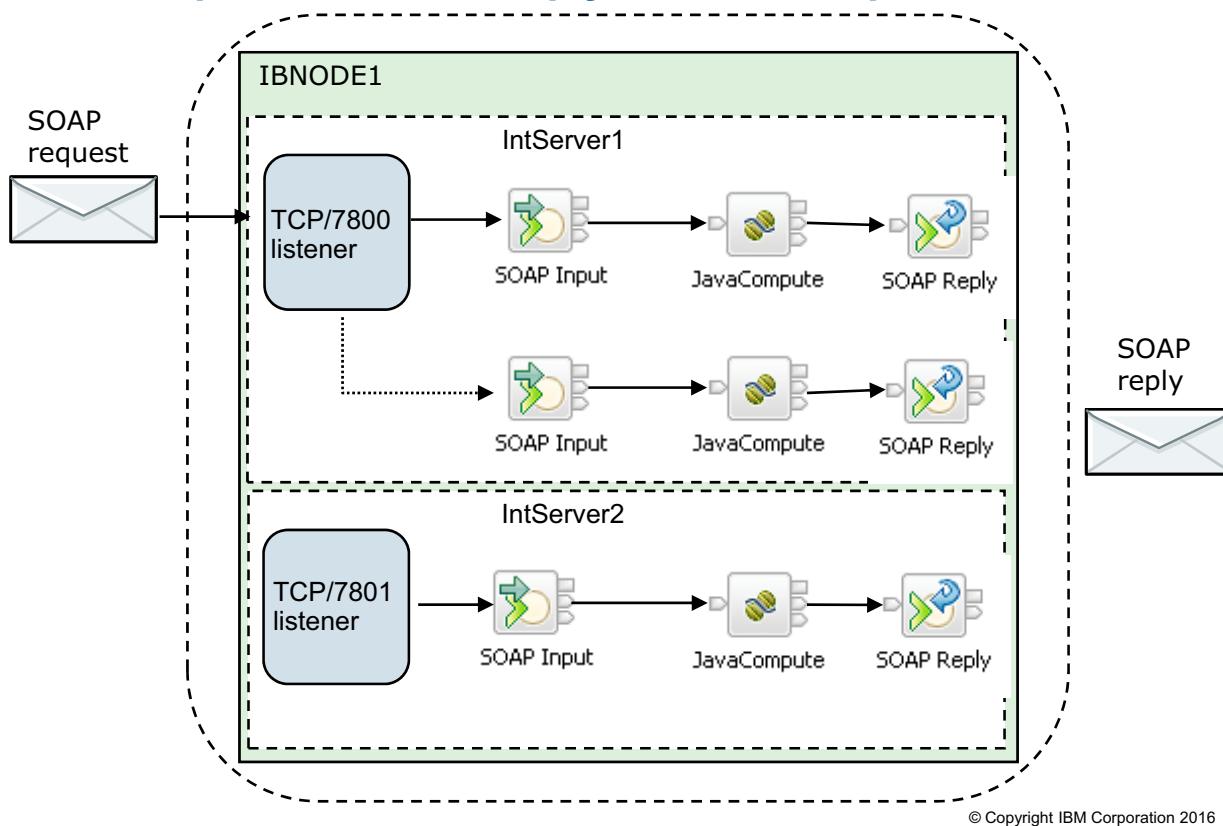
The SOAP Input and SOAP Reply nodes are analogous to the HTTP Input and HTTP Reply nodes and are used in a message flow that implements a web service. These SOAP nodes are used to construct a message flow that implements a web service provider. The SOAP Input node listens for incoming web service requests, and the SOAP Reply sends responses back to the client.

Currently, SOAP nodes support only SOAP/HTTP and SOAP/HTTPS transports. SOAP/MQ and SOAP/JMS transport support are introduced later.

SOAP nodes support WS-Addressing and WS-Security.

- WS-Addressing is a W3C specification that aids interoperability between web services by defining a standard way to address web services and provide addressing information in messages.
- WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

## SOAP Input and SOAP Reply nodes example



© Copyright IBM Corporation 2016

Figure 7-14. SOAP Input and SOAP Reply nodes example

WM676/ZM6761.0

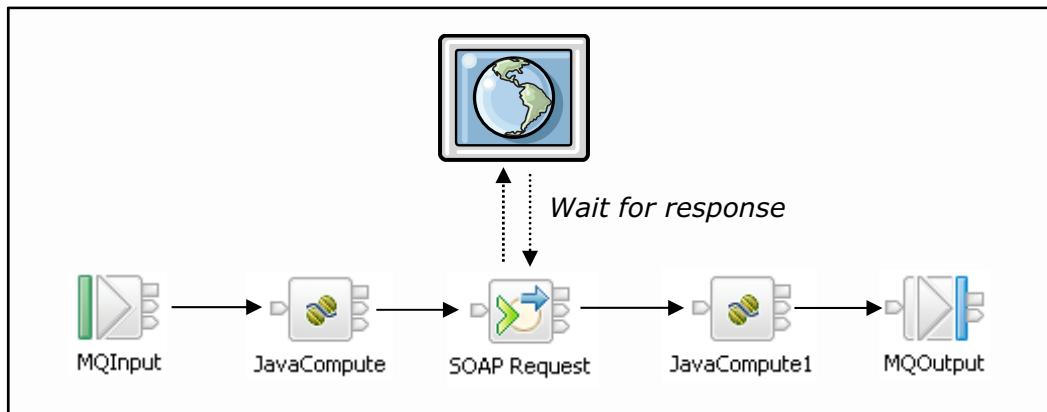
### Notes:

The figure shows an example of message flows by using SOAP Input and SOAP Reply nodes.

The HTTPSConnector object controls the runtime properties that affect the handling of HTTPS messages. Each connector has its own assigned port, which is allocated from a range of numbers, as required. The default range for the integration server HTTPConnector is 7800 - 7842; the default range for the HTTPSConnector is 7843 - 7884. The first integration server to start an embedded listener is allocated port 7800. The second is allocated 7801.

## SOAP Request node

- Synchronous request node scenario
  - Node blocks until response is received
  - Not optimal for high latency provider scenarios
  - Overall throughput is gated on latency of message response



© Copyright IBM Corporation 2016

Figure 7-15. SOAP Request node

WM676/ZM6761.0

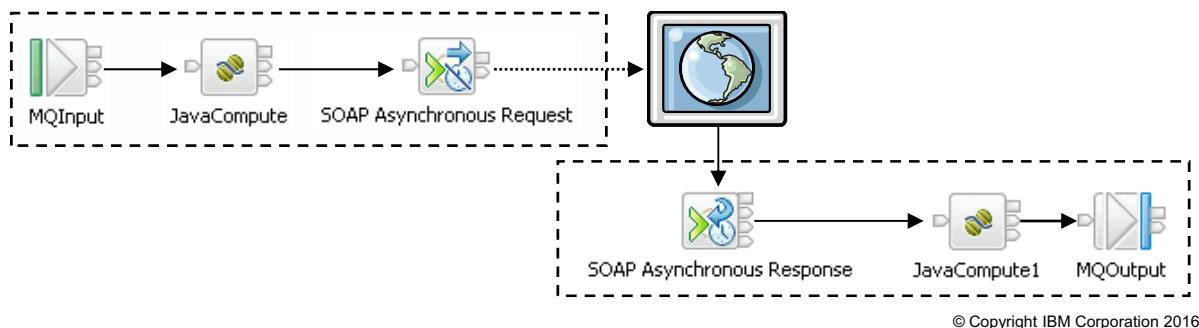
### Notes:

The SOAP Request node is used to start a web service from within a message flow (client mode).

If the web service provider potentially introduces a high latency (that is, it takes a long time to return the message), then the message flow is blocked until a response is received. This blocking can affect the overall throughput of messages through the flow.

## SOAP AsyncRequest and SOAP AsyncResponse

- SOAP AsyncRequest issues request and continues processing
  - Thread is released after request is sent
  - Does not block waiting for a response
  - Does wait for the transport acknowledgment
- SOAP AsyncResponse node waits for responses
  - Responses might be in different order to requests
  - Does not need to be in the same flow, but must be in the same integration server
- Node **Unique identifier** value identifies logical pairing of asynchronous request and response nodes
- Can handle multiple requests in parallel



© Copyright IBM Corporation 2016

Figure 7-16. SOAP AsyncRequest and SOAP AsyncResponse

WM676/ZM6761.0

### Notes:

The SOAP AsyncResponse node waits for response messages that are related to the request messages that originate from the paired SOAP AsyncRequest node. Request and reply are correlated automatically by using the data that you provide. WS-Addressing does this correlation.

## Configuring SOAP AsyncRequest nodes

© Copyright IBM Corporation 2016

Figure 7-17. Configuring SOAP AsyncRequest nodes

WM676/ZM6761.0

### Notes:

Configuring the SOAP AsyncRequest node:

- URL of the web service provider is supplied by WSDL (as is done with SOAP Input node), but can be overridden at run time with  
`LocalEnvironment.SOAP.Request.Transport.WebServiceURL`  
 (Useful in a number of environments, including when you use WebSphere Service Registry and Repository, where it can supply the URL)
- Can be configured to send requests through a proxy, dependent upon the firewall configuration
- SOAP asynchronous nodes contain a unique identifier (shared key) to pair nodes. You can store a single piece of context data in the local environment as a BLOB  
`LocalEnvironment.Destination.SOAP.Request.UserContext`

The SOAP address property of the “selected service” is used to populate the relevant information on the **Transport** tab that the “selected binding” enabled. For HTTP, the **UrlSelector** property is populated from the address information in the WSDL.

## WSDL importer

- Creates a message model from WSDL
  - Models all messages that can exist inside SOAP <Envelope>
  - Imports the predefined message definitions for the <Envelope> itself
- Accepts a range of WSDL formats
  - Single and multifile formats
  - RPC-encoded, RPC-literal, and document-literal WSDL styles
  - WSDL V1.1, SOAP V1.1, and SOAP V1.2
- Validates WSDL against WS-I Basic Profile
- Line command: `mqsicreatemsgdefsfromwsdl` or `mqsicreatemsgdefs`
- To implement the WSDL binding and endpoint details in a message flow, drag WSDL into the Message Flow editor
- Creates and configures subflows
  - To call a web service or to expose the flow as a web service
  - To wrap or unwrap the SOAP envelope

© Copyright IBM Corporation 2016

Figure 7-18. WSDL importer

WM676/ZM6761.0

### Notes:

When you import a WSDL document into Integration Toolkit, you can build a web service or web service client that is based on the operations and message model that is defined in the WSDL document.

You can import WSDL in the IBM Integration Toolkit or by using the `mqsicreatemsgdefsfromwsdl` or `mqsicreatemsgdefs` line commands.

WSDL that is copied directly into the message set folder does not contain the corresponding message definitions for the binding, so it is not usable.

WSDL is validated against the WS-I Basic Profile.

The WSDL importer creates a message model from WSDL. It models all messages that can be used inside SOAP <Envelope> and imports the IBM supplied message definitions for the <Envelope> itself.

## Import WSDL into workspace

- Import a WSDL document into an Integration Bus application, library, or integration service:
  - From the Integration Toolkit menu, click **New > Message Model**.
  - Click **SOAP XML** as the message model type.
  - In the wizard, click **I already have WSDL for my data**.

As an option, you can specify an XML schema file for the WSDL document XML schema types.

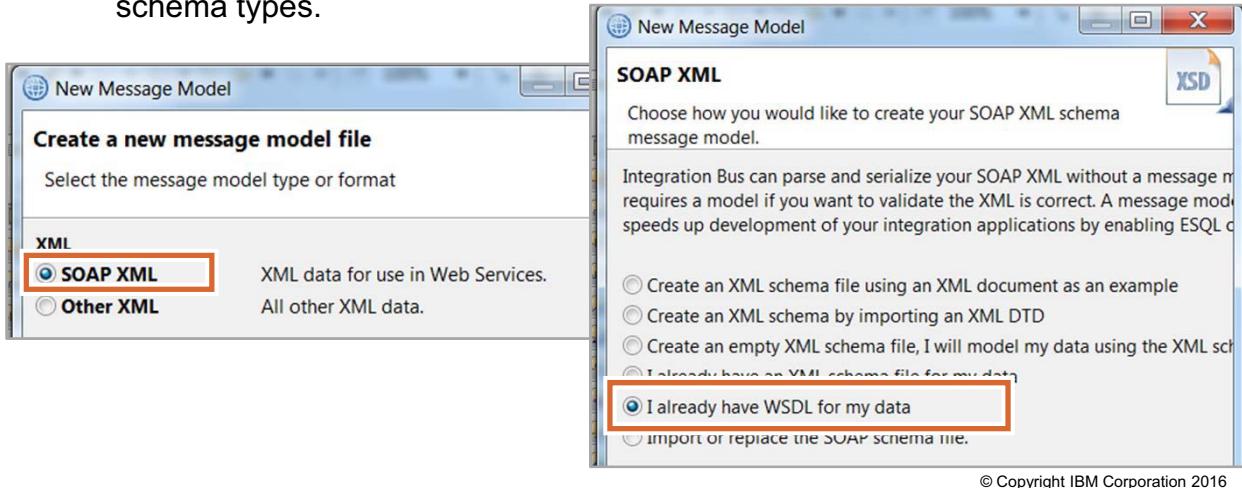


Figure 7-19. Import WSDL into workspace

WM676/ZM6761.0

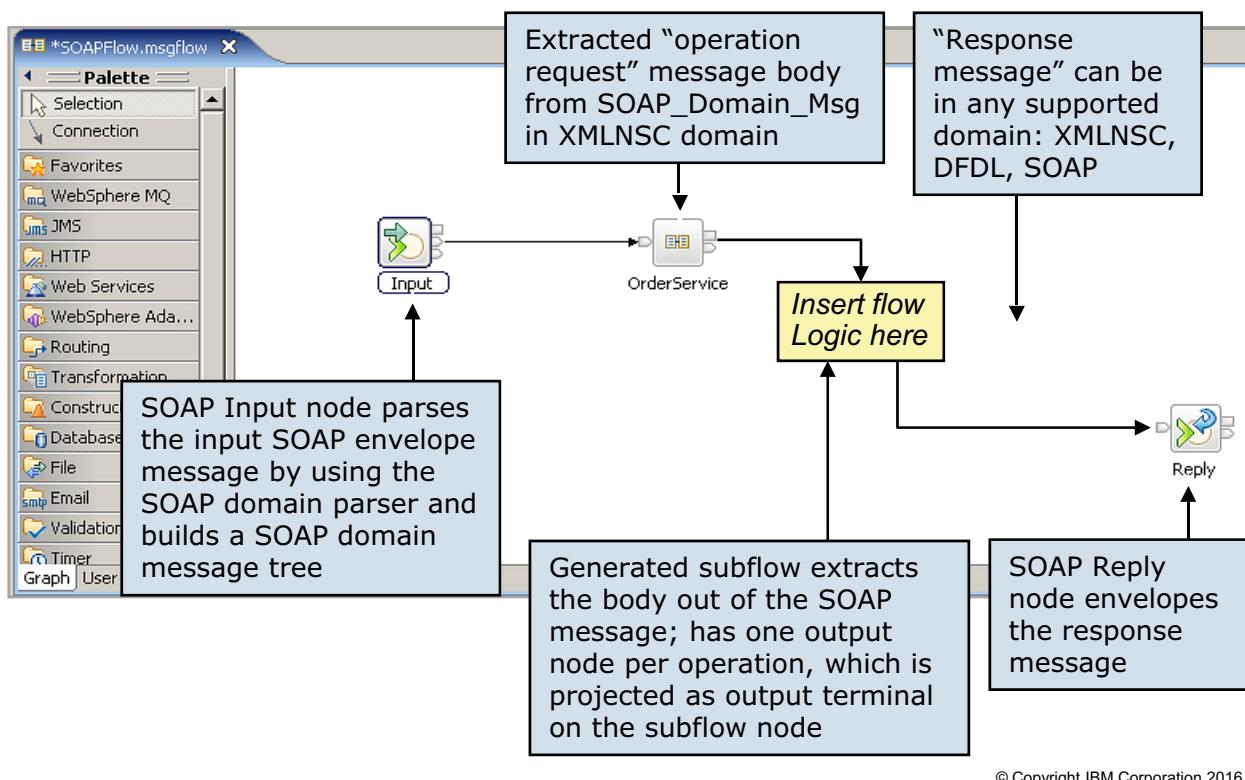
### Notes:

In a top down scenario, you build a web service or web service client that is based on an existing web service interface. A WSDL document defines the operations and data models for a web service interface. Use the message model wizard to import the WSDL document into an application, a library, or an integration service.

You can import WSDL documents into an Integration Toolkit application, library, or integration service with the **Message Model Import wizard**. Use the **SOAP XML** option as a starting point for a new message model. In the SOAP XML message model options, click **I already have a WSDL for my data** to import the WSDL document from your computer.

The WSDL document has two uses in a library or application: configuring web service operations, and working with SOAP web service message models.

## Expose a web service: Generated message flow



© Copyright IBM Corporation 2016

Figure 7-20. Expose a web service: Generated message flow

WM676/ZM6761.0

### Notes:

The message flow skeleton for SOAP nodes is generated automatically. For this generation to occur, the appropriate WSDL must be imported into the application or library. To create the message flow, you first drag the WSDL onto the flow canvas. In the wizard, you specify whether the flow should be exposed as a web service, or whether the WSDL should be used to start a remote web service. Depending on which option you select, the wizard automatically selects the binding operations and the service port that should be used. If multiple bindings and ports are available, these binding types are all listed; you select the appropriate one.

In the next stage, the wizard creates a skeleton message flow, with a number of nodes. The example on this figure shows a message flow that is exposed as a web service.

The first node is a SOAP input node that is named **Input** by default. The properties of the input node are populated automatically, by using the values that are derived from the imported WSDL, and the selections that were made in earlier stages of the wizard. Hence, the wizard generates the port type, binding, and service port. The SOAP input node creates a SOAP domain message that is based on the SOAP messages that arrive at this input node.

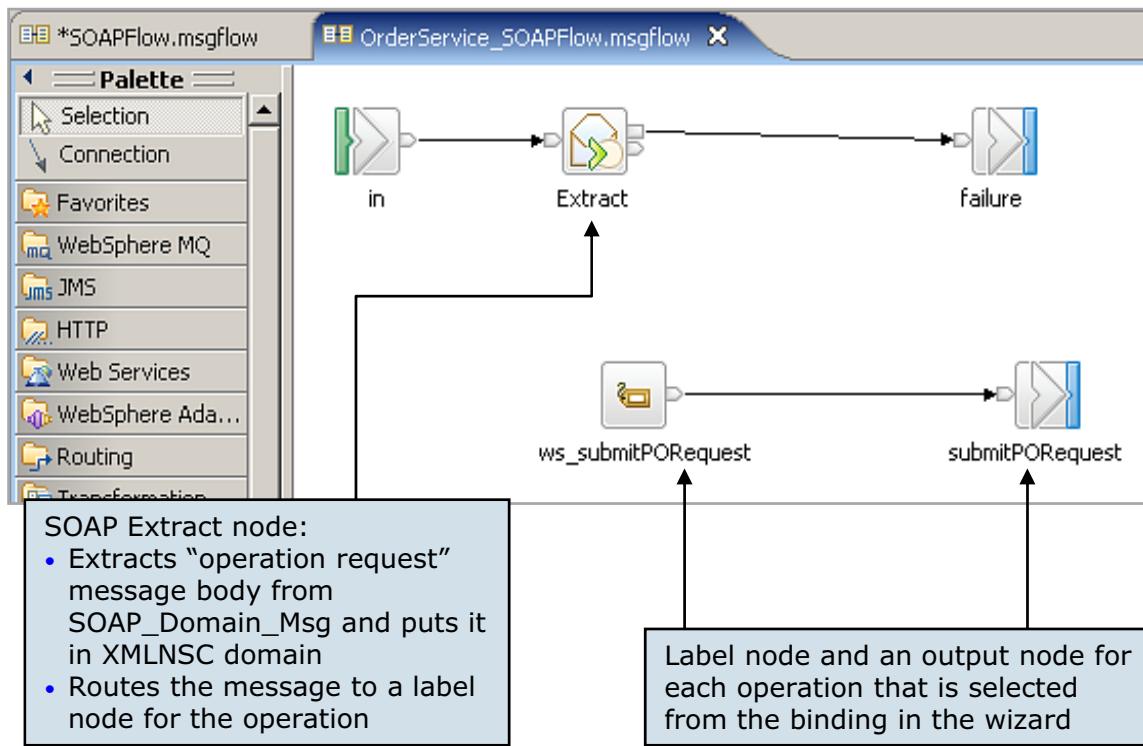
The second node, called **OrderService**, starts a subflow. This subflow exposes a terminal for each of the operations that were selected at the previous stage of the wizard. This subflow extracts the

message body and removes the SOAP envelope. The output from this subflow is the payload of the message in the XMLNSC domain.

The final node is a SOAP Reply node, which sends a response message that corresponds to the SOAP Input node.

User flow logic is placed between the generated subflow and the final Reply node.

## Expose a web service: Generated subflow



© Copyright IBM Corporation 2016

Figure 7-21. Expose a web service: Generated subflow

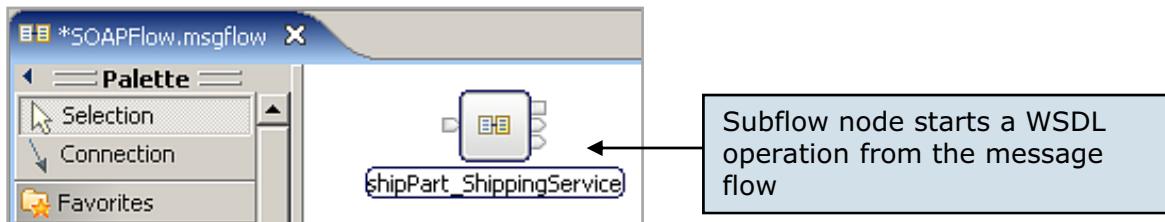
WM676/ZM6761.0

### Notes:

This figure shows the generated subflow that is used to handle the incoming SOAP message. The subflow uses a `RouteToLabel` node to handle each operation that was specified in the wizard. The example on this figure specifies only one operation; therefore, it has only one `Label` node. The `Label` node then passes the message to a corresponding `Output` node, which in turn corresponds to the appropriate output terminal on the subflow node.

The node that is named `Extract` is a SOAP Extract node that removes the SOAP envelope and places the resulting payload of the incoming message into the `XMLNSC` domain.

## Invoke a web service: Generated message flow



© Copyright IBM Corporation 2016

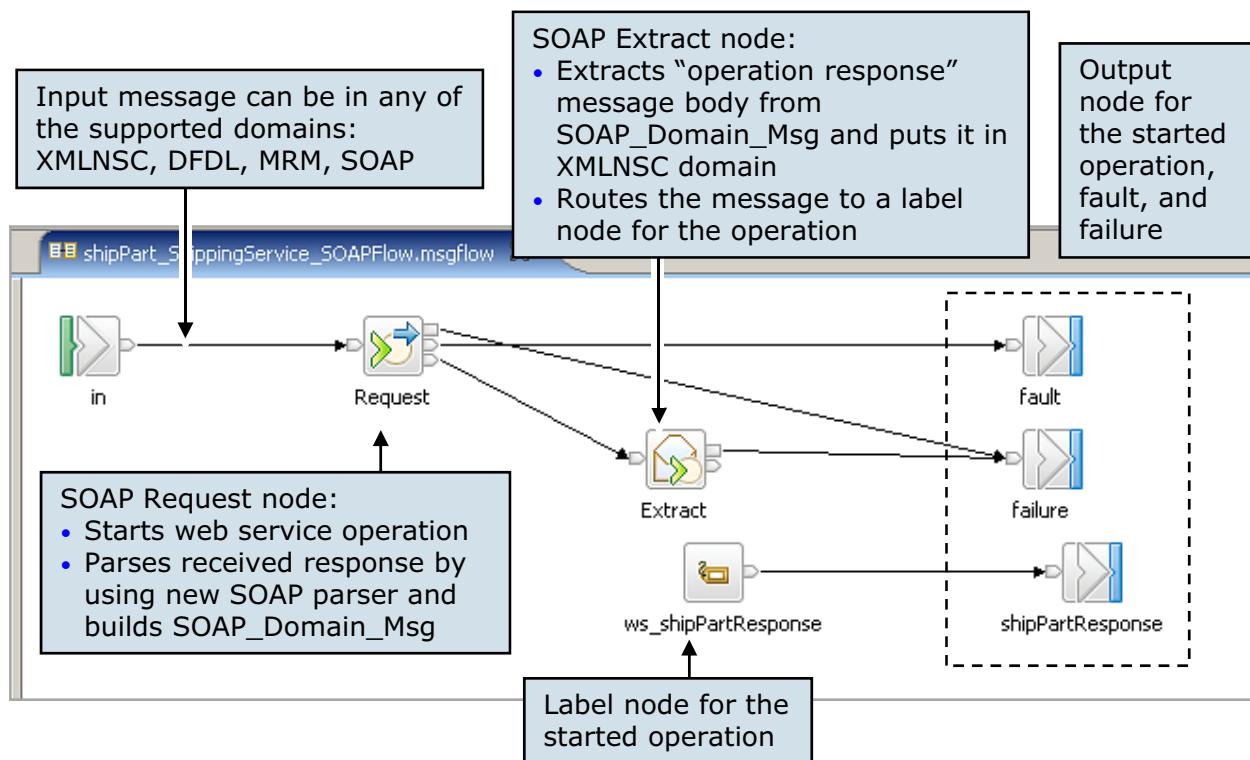
Figure 7-22. Invoke a web service: Generated message flow

WM676/ZM6761.0

### Notes:

This figure shows the client scenario, where the message flow starts a web service. This scenario generates a subflow that is named *shipPart Shipping Service*.

## Invoke a web service: Generated subflow



© Copyright IBM Corporation 2016

Figure 7-23. Invoke a web service: Generated subflow

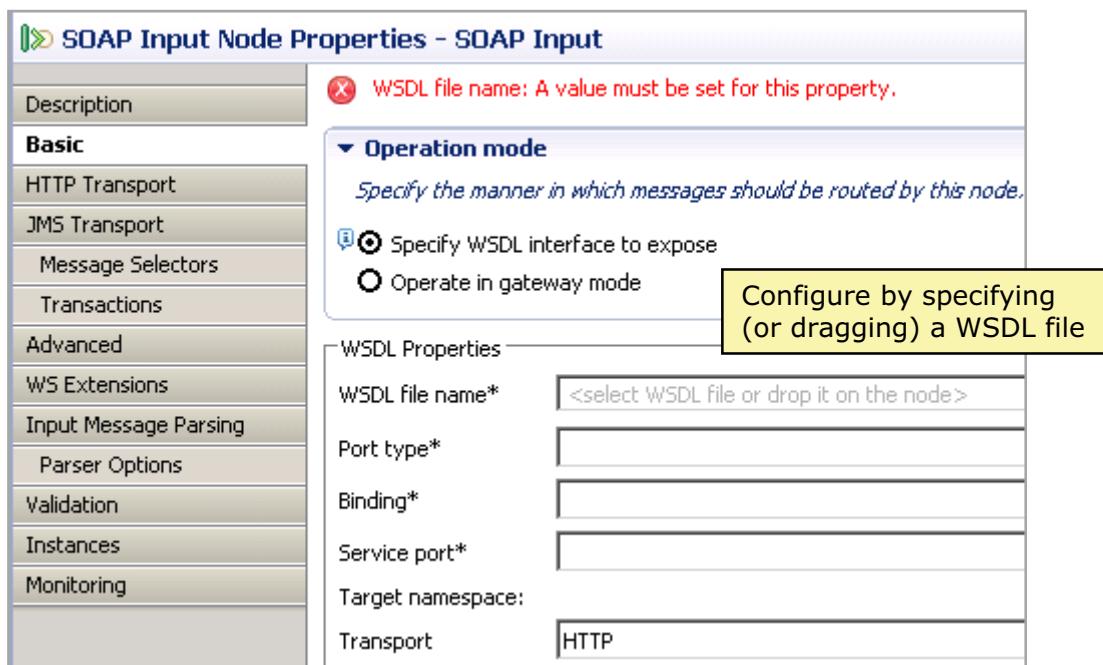
WM676/ZM6761.0

### Notes:

Expanding this subflow shows a SOAP request node. The default name of this node is Request. On completion, this node passes a message in the SOAP domain to a SOAP Extract node, which removes the SOAP envelope and passes the payload of the SOAP message to the rest of the message flow. The output from the Extract node is in the XMLNSC domain.

In this scenario, it has one label node only, along with output nodes to handle SOAP faults and failures.

## Configuring SOAP nodes



© Copyright IBM Corporation 2016

Figure 7-24. Configuring SOAP nodes

WM676/ZM6761.0

### Notes:

You can configure SOAP nodes by using a WSDL file. This approach is suggested because it reduces the possibility of configuration errors.

When you first drop a SOAP Input node on the canvas, you see the workbench provides a hint in the **WSDL file name** property: *select WSDL file or drop it on the node*. Either specify the name of the WSDL file on the **Basic** properties tab, or drag the WSDL file onto the SOAP Input node in the drawing canvas.

If you use a corresponding SOAP Reply node in the message flow, its configuration is taken from the associated SOAP Input node. A reference to the originating SOAP node is passed in the LocalEnvironment folder of the message assembly.

The properties of the SOAP Input are organized under the following tabs:

- **Basic:** For configuring WSDL properties, in particular, the WSDL file name, port type, binding, and service port. If the WSDL contains more than one binding, port Type, or other properties, then the correct one can be defined from a menu. At run time, if the SOAP message contains an operation that is not defined within the WSDL, a SOAP fault is returned. Multifile WSDL is also

supported, with the WSDL split into different files. The file that is dropped onto the node must contain the Binding or Service part of the WSDL.

- **HTTP Transport:** For the configuration of the URL Selector, use HTTPS and Maximum Client wait time.
- **JMS Transport:** You specify these properties when the messages are sent by using SOAP over JMS as the transport protocol. When you specify the *JMS provider name*, the *Initial Context Factory* is set for you automatically, but you can override the value if necessary.
- **Advanced:** SOAP role, Set destination list, Label prefix, and User-defined headers.
- **Error Handling:** If a situation arises during inbound SOAP processing that results in a SOAP fault, send the SOAP fault to the Failure terminal to allow logging and recovery processing, rather than returning it to the client. In this situation, an exception list is sent down the Failure terminal with the inbound message as a BLOB. The default value of this property is false.
- **WS-Extensions cover WS-Addressing and WS-Security (XPath):** XPath expressions that have an associated alias value are added to the WS-Security table. The alias is resolved in a policy set that the administrator creates. The policy set resolves the alias to either encrypt or sign the part of the message referred to by the XPath expression.

## Generating WSDL from message set

- In the Integration Toolkit, right-click the folder that contains the message set file and then select **Generate > WSDL Definition**
  - Generate a new WSDL definition from existing message definitions
  - Export an existing WSDL definition to another directory in the workspace or file system
- One or more bindings can be requested
  - SOAP/JMS, SOAP/HTTP, JMS TextMessage
- WSDL message types match MRM logical messages
  - Physical representations can require mapping or transformation for bindings
- Supports both the recognized WSDL styles: “rpc” and “document”
- Only “literal” encoding; SOAP encoding not supported
- Validates generated WSDL against WS-I Basic Profile

© Copyright IBM Corporation 2016

Figure 7-25. Generating WSDL from message set

WM676/ZM6761.0

### Notes:

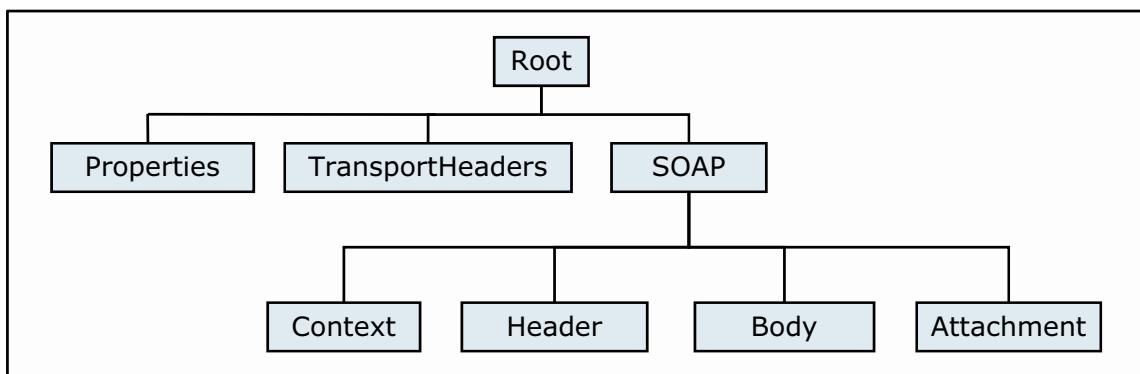
If an integration node is to communicate with a web service client, it typically needs to accept SOAP messages. One approach is to use the MRM domain, in which case the message model and the WSDL definition that the web service client uses must describe the same messages.

If the integration node has an existing message model (for example, created by importing a C header file or COBOL copybook), it can be exported to create a corresponding WSDL definition for use by the client. At the same time, your message model needs to be enhanced with appropriate definitions for the SOAP envelope and the WSDL operations (for RPC-style).

If the integration node must interact with an existing web service, you can import a WSDL definition into a message set. The resulting message set contains message definitions that model the SOAP envelope and the content of the corresponding SOAP messages. A flow developer can use these definitions to validate and work with an incoming message; for example, defining a mapping to transform a SOAP request message into a SOAP response message. Various WSDL styles are accepted.

## Common SOAP message tree

- SOAP domain and parser offer a consistent approach for constructing a flow to handle web services, regardless of the specific bitstream format
- WSDL document triggers SOAP parser events and is used to validate the SOAP envelope
- Web service message can be SOAP, SwA, or MTOM
- Context contains WSDL-related information such as operation name
- Generated from the WSDL



© Copyright IBM Corporation 2016

Figure 7-26. Common SOAP message tree

WM676/ZM6761.0

### Notes:

The SOAP parser represents web service messages (requests, responses, and faults) with the same logical tree shape irrespective of the specific bitstream format (SOAP or SwA/MIME). The SOAPIInput node generates a tree whose shape is unaffected by the input message. By contrast, the message assembly that an HTTPInput node produces differs depending on whether the message was MIME format or SOAP format.

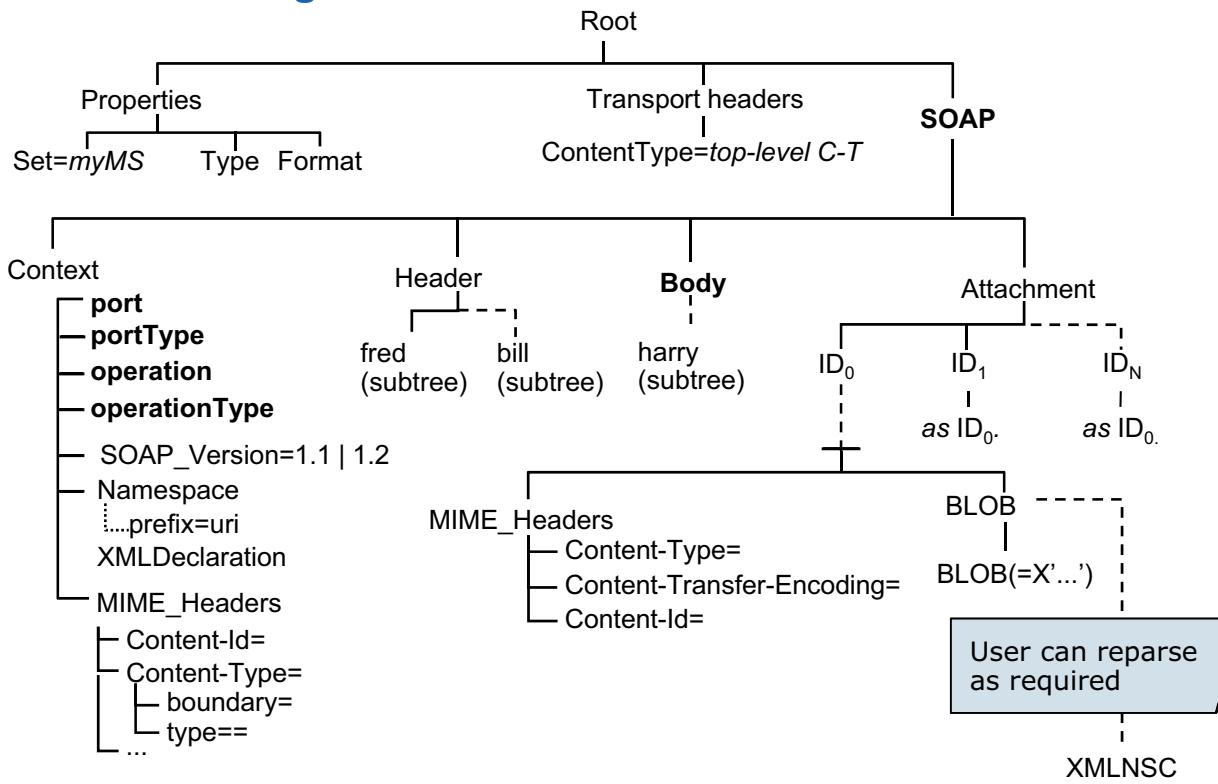
WSDL is deployed to the integration node and used to validate the SOAP messages that are received (for the SOAPIInput node). For example, you can ensure that the operation that is received in the incoming SOAP message is defined within the WSDL.

The bitstream format for these runtime messages can be SOAP V1.1 or SOAP V1.2, and optionally wrapped by MIME as a SOAP with Attachments (SwA) or MTOM message.

The following example WSDL illustrates a simple operation that has one attachment that is called "attach":

```
<binding name="MyBinding" type="tns:abc" >
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="MyOperation">
<soap:operation soapAction="" />
<input>
<mime:multipartRelated>
<mime:part>
<soap:body parts="part1 part2 ..." use="encoded" namespace="http://mynamespace"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding" />
</mime:part>
<mime:part>
<mime:content part="attach" type="text/html" />
</mime:part>
</mime:multipartRelated>
</input>
</operation>
</binding>
```

## SOAP message tree



© Copyright IBM Corporation 2016

Figure 7-27. SOAP message tree

WM676/ZM6761.0

### Notes:

This figure shows a more detailed view of the SOAP message tree.

The **SOAP.Context** branch contains certain information that the SOAP envelope contains. It also shows the SOAP version that is being used. The SOAP parser sets the following information (derived from the WSDL) under **SOAP.Context** on input: **port**, **portType**, **operation**, **operationType**.

The **Content-Id** within the **Attachment** section was copied directly under the **Attachment** part, allowing the specific message part to be easily identified and parsed.

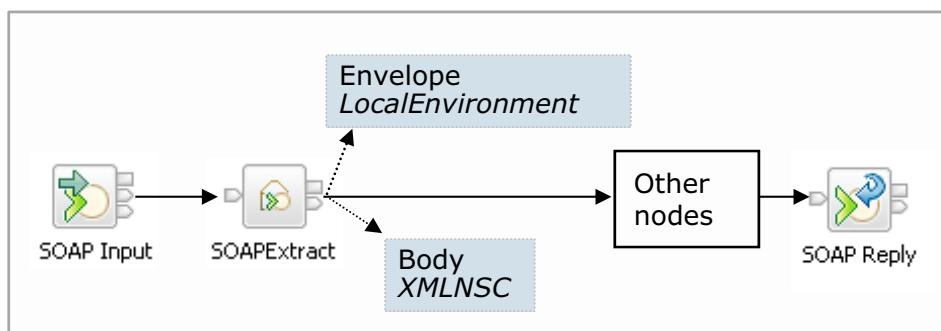
The SOAP parser represents web service messages (requests, responses, and faults) with the same logical tree shape irrespective of the specific bitstream format (SOAP or SwA/MIME).

The SOAP payload (under **Envelope.Body**) is saved under **SOAP.Body**.

The SOAP header blocks (under **Envelope.Header**) are saved under **SOAP.Header**.

## SOAP Extract node

- Allows the user to work on the payload of the SOAP body
  - Context, headers, and any attachments are removed and stored in the LocalEnvironment for possible later use
  - These elements are automatically picked up and restored when a SOAP Reply node is encountered in the flow



© Copyright IBM Corporation 2016

Figure 7-28. SOAP Extract node

WM676/ZM6761.0

### Notes:

The SOAP Extract and SOAP Envelope nodes work on the payload of the SOAP body. The SOAP Extract node can interoperate with the SOAP domain.

The SOAP Extract node can do two functions:

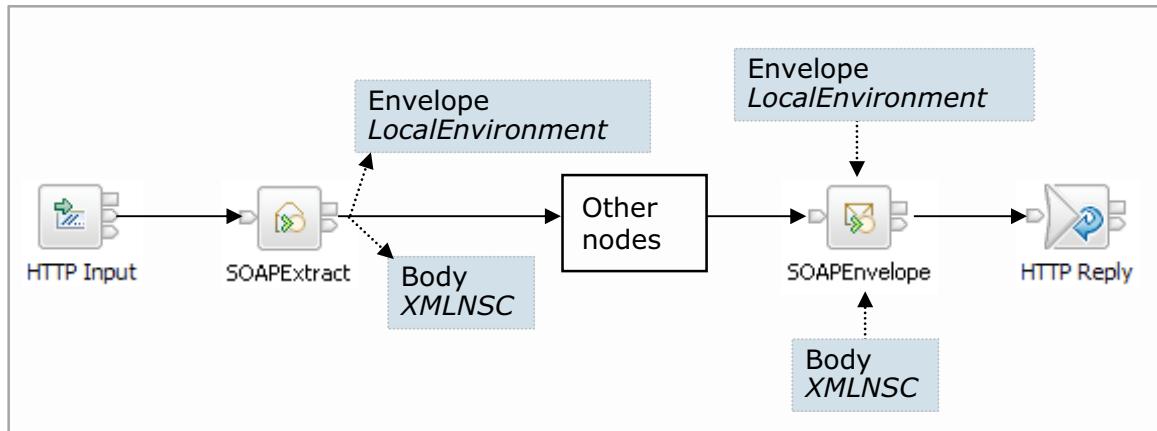
- Extract: The default behavior is to detach the SOAP envelope to a standard location (`$LocalEnvironment/SOAP/Envelope`) in the LocalEnvironment tree. Alternatively, you can specify an explicit location by using an XPath expression. Any existing SOAP envelope at the chosen location is replaced.
- Routing: The SOAP message is routed to a Label node within the message flow that the SOAP operation identifies within the message. The SOAP operation is identified within the SOAP body tag.

Both functions are optional; they are contained within one node because they are often used together.

The SOAP Envelope node is required for HTTP nodes, but not for SOAP nodes.

## SOAP Envelope node

- SOAP Envelope node can be used to insert SOAP headers into a non-SOAP message
- SOAP nodes do not require the SOAP Envelope node
  - Can directly handle non-SOAP messages (and look at the LocalEnvironment)



© Copyright IBM Corporation 2016

Figure 7-29. SOAP Envelope node

WM676/ZM6761.0

### Notes:

SOAP nodes do not require the SOAP Envelope node because they can directly handle non-SOAP messages (and look at the LocalEnvironment). However, the SOAP Envelope node is still required for the HTTP nodes.

The default behavior of the SOAP Envelope node is to attach the SOAP envelope from a standard location (`$LocalEnvironment/SOAP/Envelope`) in the LocalEnvironment tree. You can specify an explicit location by using an XPath expression. You can also use the node in a flow without a corresponding SOAP Extract node. The node can optionally create a default SOAP envelope.

## 7.4. WS-Addressing and WS-Security

## WS-Addressing

- WS-Addressing is a standardized way of including the addressing data in the SOAP message itself
  - Endpoint reference (EPR) is the information that is needed to address a web service endpoint
  - Message addressing properties (MAPs) contains a list of addressing properties such as ReplyTo, FaultTo, MessageID, and Action
- Integration Bus automatically supports WS-Addressing
  - SOAP nodes: Use **WS-Addressing** check box
  - WSA headers flow in LocalEnvironment
  - Large number of overrides and status information is contained within the LocalEnvironment

© Copyright IBM Corporation 2016

Figure 7-30. WS-Addressing

WM676/ZM6761.0

### Notes:

SOAP messages are generally transmitted over HTTP, and reply on the HTTP URL to ensure that the message reaches the intended destination.

The HTTP address is also used to ensure that the response is sent back to the originator of the message. HTTP does not contain any mechanism that allows the originator of the message to indicate that the reply should be sent to a different destination, rather than back to the originator. By using the WS-Addressing standard, you can set this alternative address, instead of using a custom technique. You set the alternative address by defining WS-Addressing headers, which contain information that describes where the reply to the message should be sent.

To set an alternative address, WS-Addressing defines two constructs: endpoint reference and the message addressing properties.

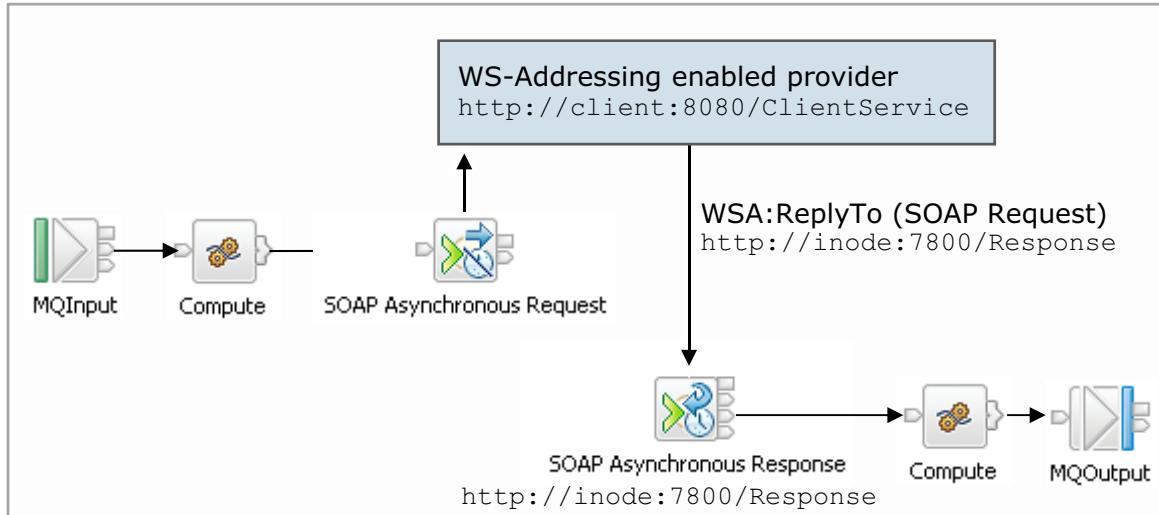
- The endpoint reference contains information that is needed to address a web service endpoint.
- The message addressing properties contain a number of properties that allow different parts of the message to be addressed properly. These properties include items such as the “ReplyTo” address and the “FaultTo” address, which specifies where faults should be directed.

In Integration Bus, WS-Addressing must be explicitly enabled on the SOAP Input node properties. No further logic or work is required in the message flow. However, if you need to manipulate addressing information in the flow, you can do so. WS-Addressing information is passed into the flow by using the LocalEnvironment. Most of the WS-Addressing fields can be overridden by setting the appropriate fields in LocalEnvironment, for example,

`LocalEnvironment.Destination.SOAP.Reply.Transport.HTTP.ProxyURL`. See the documentation for details.

If it is not enabled, any responses (including faults) are returned to the client. WS-Addressing headers of the inbound SOAP message are ignored. If these headers are marked `mustUnderstand`, then a fault is generated.

## WS-Addressing asynchronous consumer scenario



- WS-Addressing is enabled by default and cannot be turned off
- SOAP AsyncRequest node places the return address of the associated SOAP AsyncResponse node into the **WSA:ReplyTo** fields

© Copyright IBM Corporation 2016

Figure 7-31. WS-Addressing asynchronous consumer scenario

WM676/ZM6761.0

### Notes:

In this scenario, the message flow uses the SOAP Asynchronous Request node to start a web service. The response to the original web service request is handled separately from the request. In this case, the use of WS-Addressing is mandatory, and the property is automatically set on the SOAP Asynchronous Request node.

The SOAP Asynchronous Request node generates the required WS-Addressing headers. However, the “Reply To” header is the reply address of the associated SOAP Asynchronous response node.

Because the web service provider is enabled for WS-Addressing, the reply recognizes the “Reply To” header, and is sent back to the SOAP Asynchronous Response node. The second part of the message flow then completes as normal.

The SOAP fault conditions are understood, and faults are sent to the address in the “Fault To” header when it is specified.

## What is WS-Security?

- WS-Security specification defines how to store security metadata in the SOAP message header
- Examples of security metadata:
  - Authentication credentials
  - Security assertions that an intermediary assigns
  - A digital signature for a part or the entire SOAP message
  - A reference to the security certificate that the server uses to decrypt a part or the entire SOAP message body
  - Security audit information
- WS-Security framework defines security profiles to support existing security standards

© Copyright IBM Corporation 2016

Figure 7-32. What is WS-Security?

WM676/ZM6761.0

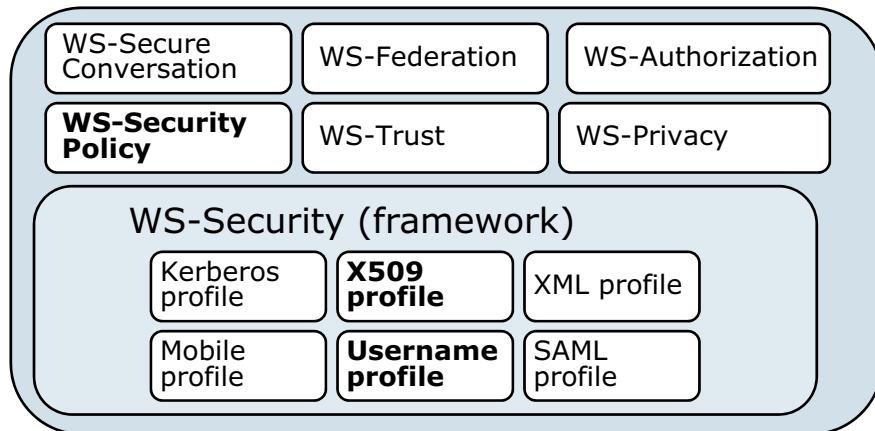
### Notes:

IBM Integration Bus focuses on the part of the WS-Security specification that is known as WS-Security Policy. This policy is implemented by using the properties of the SOAPInput or SOAPRequest Nodes.

Key areas that IBM Integration Bus encompasses include authentication; message-level protection through digital signature, encryption, or both; and message part protection through digital signature, encryption, or both.

## Web services and security

<http://oasis-open.org/committees/download.php/1204/doc-index.html>



© Copyright IBM Corporation 2016

Figure 7-33. Web services and security

WM676/ZM6761.0

### Notes:

What is often thought of as a single specification, WS-Security, is a large set of interrelated specifications. The figure shows the components that constitute WS-Security. It is important to note that the WS-Security specification is large, and many software vendors do not implement the entire specification. A software component normally implements the security features that are appropriate for the type of security usage that is required.

## Transport level security and message security

- Transport level security (SSL and HTTPS)
  - Protects the stream of data that is being passed from one endpoint to another
  - Typically all of the data is encrypted; does not discriminate on a per message basis (everything is encrypted) with the same key
  - When passing messages by using another intermediary, if the intermediary must “see” any part of the message, it can access all of it
- Message level security (WS-Security)
  - Message-based security provides finer granularity
  - Security is applied on a per message basis
  - Parts of a message can be (multiply) encrypted and signed on a “need-to-know” basis
  - WS-Security can be used with insecure transports

© Copyright IBM Corporation 2016

Figure 7-34. Transport level security and message security

WM676/ZM6761.0

### Notes:

Security and SOAP messages have many elements. The task is to keep the message secure through intermediate systems until it reaches the ultimate recipient.

WS-Security is a message-level standard that is based on securing SOAP messages through an XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. The web services security specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. WS-Security does not require any specific type of security token. It is extensible; for example, it supports multiple security token formats.

WS-Security also describes how to encode binary security tokens and attach them to SOAP messages. Specifically, the WS-Security profile specifications describe how to encode user name tokens and X.509 tokens. With WS-Security, the domain of these mechanisms can be extended by carrying authentication information in web service requests. WS-Security also includes extensibility mechanisms that can be used to further describe the credentials that are included with a message. WS-Security is a building block that can be used with other web service protocols to address a range of application security requirements.

Using WS-Security has numerous advantages.

- Different parts of a message can be secured in various ways. For example, you can use integrity on the security token (user ID and password) and confidentiality on the SOAP message body.
- Intermediaries can be used and end-to-end message-level security can be provided through any number of intermediaries.
- WS-Security works across multiple transports and is independent of the underlying transport protocol.
- Authentication of both individual users and multiple party identities is possible.

## WS-Security <Security> element

- WS-Security specification defines a vocabulary that can be used inside the SOAP envelope
- <wsse:Security> is the “wrapper” for security-related information in the SOAP envelope header

- User name and password
- X.509 certificate
- Encryption details
- XML signature

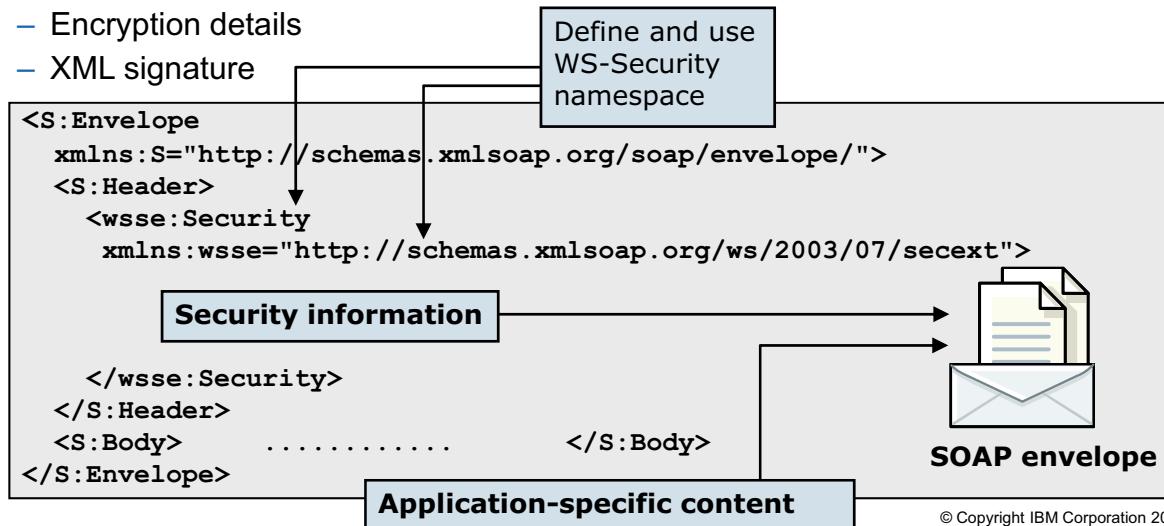


Figure 7-35. WS-Security &lt;Security&gt; element

WM676/ZM6761.0

### Notes:

A SOAP message consists of an envelope that contains the whole message. Within the envelope is the actual message, which consists of header information and a SOAP body. The body is the real payload, that is, the message that an application program typically needs to access.

WS-Security defines a new <Security> element that can be placed in the header section of a SOAP message: user name and password, X.509 certificate, encryption details, or XML signature.

The example shows a SOAP message with a skeleton showing where WS-Security should be placed. The element wsse:Security is used to contain the security information.

## WS-Security authentication in IBM Integration Bus

- User name and password (Username Token) are fully supported when using either LDAP or a WS-Trust v1.3 STS to authenticate the identity
- X.509 Certificate (Binary Token), which uses Java key and truststore configured on the integration node or integration server
- Kerberos Token Profile, which uses the JVM/Host Kerberos infrastructure, `krb5.conf`, and `keytab` files



Figure 7-36. WS-Security authentication in IBM Integration Bus

WM676/ZM6761.0

### Notes:

Web services security defines two types of security tokens: a *user name* token and a *binary* security token.

A user name token consists of a user name and optionally, password information. You can include a user name token directly as an element in the Security header within the SOAP message, rather than the HTTP header. Binary tokens, such as X.509 certificates, Kerberos tickets, Lightweight Third-Party Authentication tokens, or other non-XML formats, require a special encoding for inclusion. The user name token is used to provide a user name within the SOAP message. It is used as part of the basic authentication process.

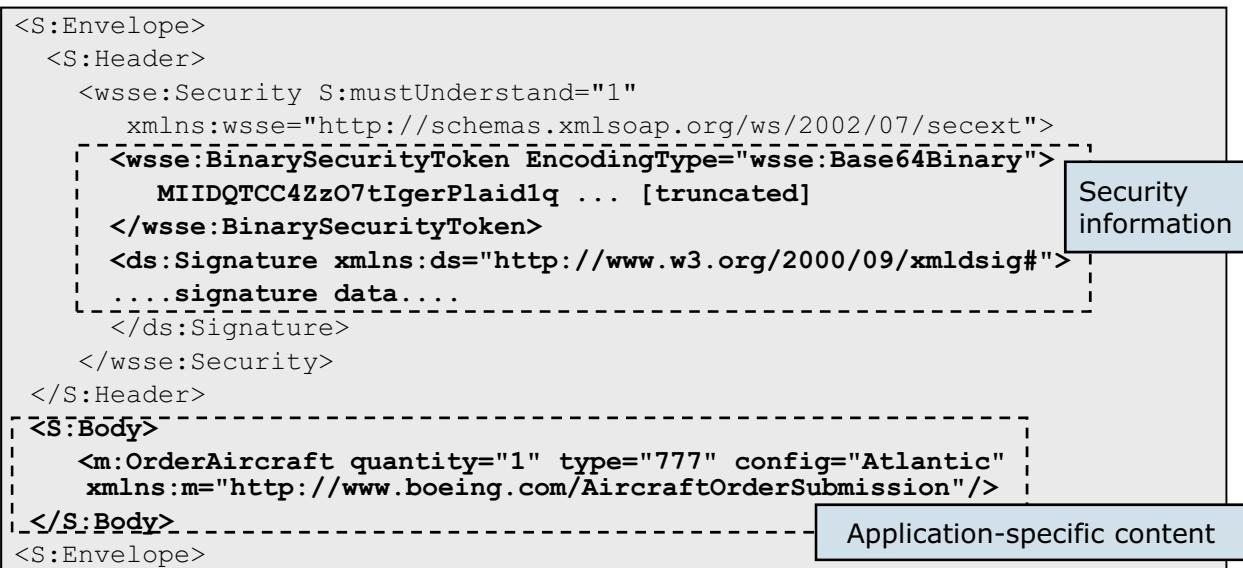
IBM Integration Bus supports binary tokens for authentication. The tokens that are supported are asymmetric X/509 tokens. Symmetric X.509 tokens are not supported.

In this example, a user name token is specified. The user name and password are specified in clear text. This text can be encrypted by using message part encryption, which is described later.

The IBM Integration Bus administrator must enable the Integration Bus security manager and specify a security profile. If these steps are not done, all user names and passwords are accepted.

## WS-Security message level protection

- Entire body within the SOAP message can be encrypted and signed
  - Using different certificates, if required
- Signed (XML signature): Message can be read but not changed
- Encrypted (XML encryption): Message cannot be read or changed



© Copyright IBM Corporation 2016

Figure 7-37. WS-Security message level protection

WM676/ZM6761.0

### Notes:

IBM Integration Bus also supports message protection. Either the entire message or part of the message can be protected.

The message can be encrypted, digitally signed, or both.

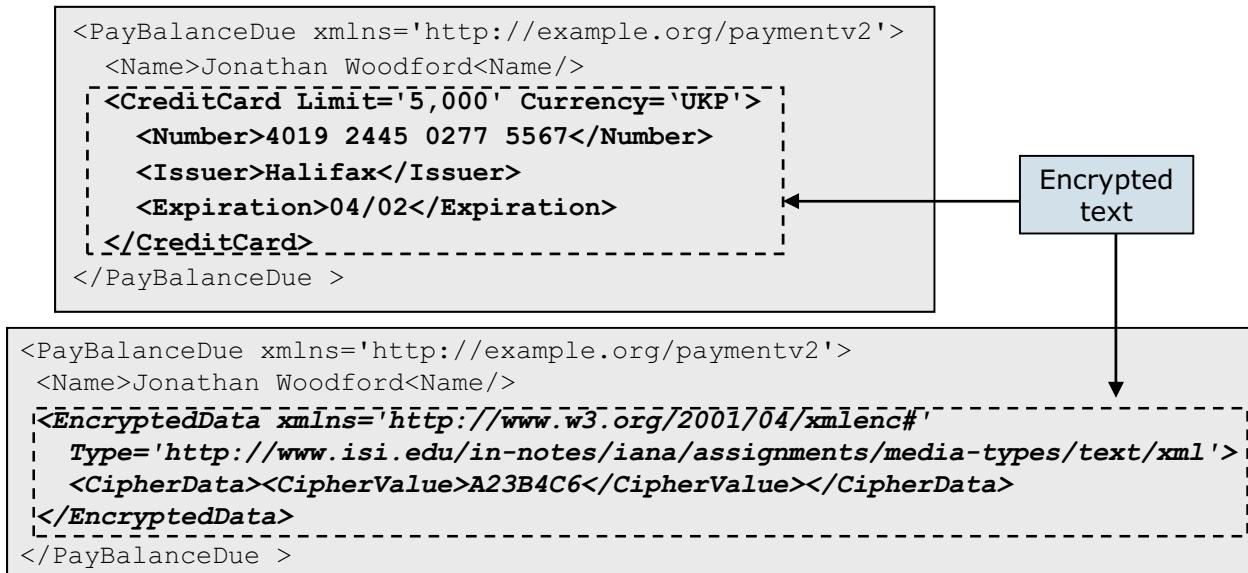
These different security functions can be used through different security certificates, and can be used separately from each other.

The example shows the payload that is being digitally signed. It is not encrypted, and is still readable by intermediaries.

By using this technique, you can allow intermediaries to have access to certain parts of the message, but inhibit access to other parts of the message.

## WS-Security message part protection

- Allows header and body to be encrypted and signed
- Different certificates can be used for different parts of the message
- Element and namespace (QNAME) support



© Copyright IBM Corporation 2016

Figure 7-38. WS-Security message part protection

WM676/ZM6761.0

### Notes:

As was described previously, it is possible to protect certain parts of the SOAP message, rather than the entire message. You protect the message parts by specifying elements and namespaces within the header or body of the SOAP message that you want to sign or encrypt. You use the QNAME function to specify namespaces.

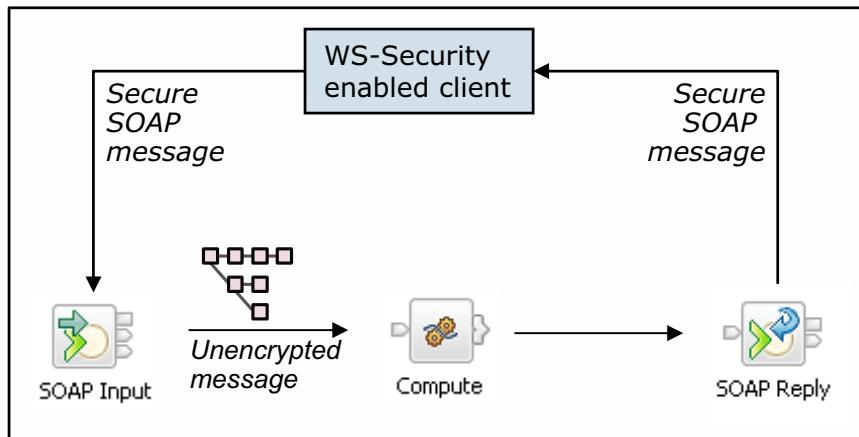
Because message part protection can specify the header, it can be used with user name tokens to protect the user name and password.

The example shows a credit card transaction, where the credit card details are encrypted. The section at the bottom of the figure contains the message that is sent over the transmission channel. The sensitive credit card data is shown as encrypted.

The encrypted information in this example is truncated.

## WS-Security: Provider scenario

- SOAP nodes handle the WS-Security function



- SOAP Input node authenticates, decrypts, and verifies the signature of the message and message parts
- Message passes unencrypted through the message flow
- SOAP Reply node encrypts and signs the appropriate parts of message

© Copyright IBM Corporation 2016

Figure 7-39. WS-Security: Provider scenario

WM676/ZM6761.0

### Notes:

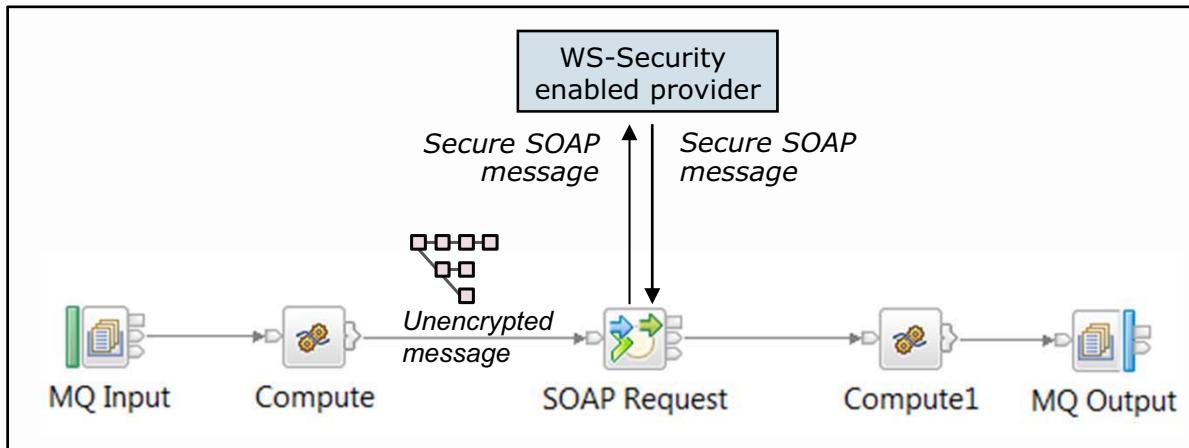
This figure shows a scenario where the message flow represents the web service. A web service client starts the message flow by sending it a SOAP message. In this case, the SOAP message is secured by using WS-Security. The SOAP nodes handle the WS-Security function. No logic or action is required in the rest of the flow.

The role of the SOAP Input node is to verify all signed parts and decrypt the message as defined within the configuration. It does so by using the security configuration that it is given. The role of the SOAP Reply node is to apply any required WS-Security function. If the WS-Security configuration is applied to the whole message, then the entire message is unencrypted for use by the message flow.

If the WS-Security configuration defines parts of the message, then the SOAP Input node decrypts these parts for use by the message flow. Any parts of the message that IBM Integration Bus is not allowed to see because it does not contain the correct certificates remain encrypted.

When the message reaches the SOAP Reply node, the message must be signed and encrypted, according to the definition of the WS-Security configuration for the message flow.

## WS-Security: Consumer scenario



- SOAP Request node encrypts and signs the appropriate parts of message
- Message is sent to the WS-Security enabled provider
- SOAP Request node decrypts and verifies the signature of the message and message parts

© Copyright IBM Corporation 2016

Figure 7-40. WS-Security: Consumer scenario

WM676/ZM6761.0

### Notes:

## Policy sets

- WS-Security within Integration Bus is configured through policy sets and policy set bindings
- Policy set is a collection of configuration information that defines what is required for Integration Bus WS-Security
  - Express what security you want to apply to the message: authentication, integrity, confidentiality
  - Which parts of the message must be signed and encrypted
- Policy set bindings define how Integration Bus WS-Security is configured
  - What keys can be used and which security policies are required
  - Contains the physical security credentials
- Policy Set editor is provided in the Integration Toolkit

© Copyright IBM Corporation 2016

Figure 7-41. Policy sets

WM676/ZM6761.0

### Notes:

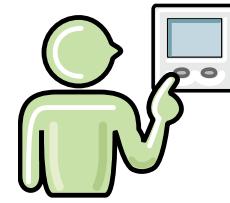
Configuration of WS-Security is achieved by using policy sets. Policy sets build on WS-Security Policy, and add a number of usability components to the base function.

A policy set refers to a set of policy types. Each policy set contains two parts. The policy set itself refers to a set of policy types. IBM Integration Bus supports only the WS-Security Policy type. The policy set defines what is needed for WS-Security; it defines the different types of authentication, integrity, and confidentiality that are needed. In addition, it defines which parts of the message need to be signed or encrypted.

The policy set bindings define how the security policies are used and applied to the defined message. It contains the physical security credentials; for example, which keys are going to be used, which certificates are going to be used, and where are these certificates stored?

## Administrative versus development responsibilities

- Policy set configuration is handled primarily as an administrative task
  - The administrator can handle most of the process
  - Authentication, body encryption, and signature



- For message part protection, configuration consists of development and administration tasks
  - Nodes allow the developer to specify elements that require message part protection applied
  - Administrator decides how the part of the message is to be encrypted or signed

© Copyright IBM Corporation 2016

Figure 7-42. Administrative versus development responsibilities

WM676/ZM6761.0

### Notes:

The implementation of the security requirements for a particular application is split between the application developer and the system administrator.

Most of the implementation is done as an administration task. For message-level protection, the administrator is able to do all the necessary tasks to implement WS-Security. However, administrators might not be familiar with the message tree contained within the SOAP header and SOAP body. If you need to implement message part protection, you can specify which parts of the message should be protected.

With the SOAP nodes, you can specify elements that require the message part protection to be applied. The policy set editor allows the administrator to specify how each of these message parts is to be encrypted or signed.

Message processing nodes allow the developer to specify elements that require message part protection to be applied. The administrator decides how the message parts are to be encrypted or signed.

## Defining message part protection on the node

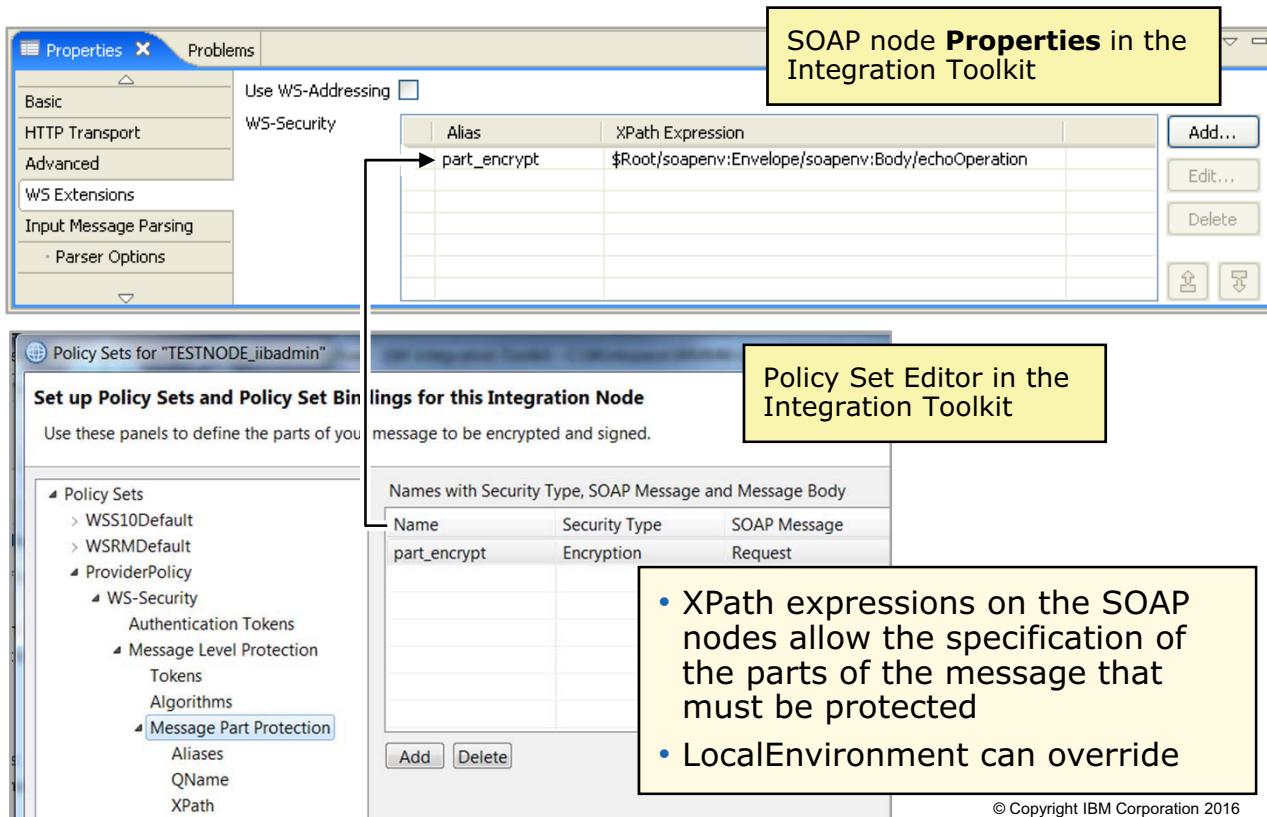


Figure 7-43. Defining message part protection on the node

WM676/ZM6761.0

### Notes:

Along with the other WS-Security settings, you can use the PolicySet Editor to configure message part protection. Although the administrator needs to be able to specify this information, developers can have a better grasp on the structure of the message (within the SOAP body) because they are directly working with it. Therefore, it is important that they have a mechanism for specifying parts of the message that have WS-Security considerations. For example, some parts need to be encrypted because they contain private data such as credit card details.

This figure shows how the properties of the SOAP nodes are used to specify security on parts of the SOAP message. On the SOAP node **WS-Extensions** property tab, an alias is added for each message part that is subject to security. This alias can be contained in the header of the message or the body of the message. You can use XPath expressions to specify the parts of the message that must be protected.

The alias provides a link between the configuration in the SOAP nodes and the runtime components that the system administrator specifies. The IBM Integration Bus administrator defines the Policy Set, which resolves the alias to either encrypt or sign the part of the message that the XPath expression references. Specifying the policy set is an administrative function. You cannot define or reference it with the SOAP node properties.

You can also use the LocalEnvironment to specify parts of the message that must be signed or encrypted. This capability can be used to change the security requirement dynamically (based on the message, for example), for different types of messages. This technique still uses the alias to link the XPath expression that is defined with the policy set.

## Assigning policy sets with flows and nodes

- Different web services can have different requirements on WS-Security
  - Different services can use different certificates
  - Different messages (for differing uses) can require different parts of the message to be encrypted
- A single policy set and policy set binding is not sufficient for all nodes in all flows
- Policy set can be assigned at the flow or node level, depending on the required granularity

© Copyright IBM Corporation 2016

Figure 7-44. Assigning policy sets with flows and nodes

WM676/ZM6761.0

### Notes:

It is possible to define more than one policy set and policy set binding for each instance of an integration node.

The message flows deployed to that integration node can have different requirements for web services security. Each service can use different certificates and have different requirements for message part processing. Therefore, it is necessary to be able to assign a policy set and binding to a specific set of web services or message flows.

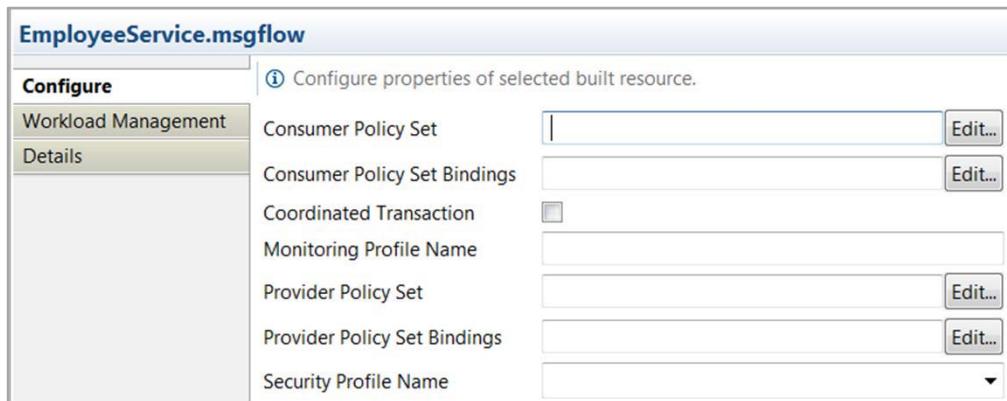
The policy set can be assigned at the flow level, or if the requirements are more granular, it can even be assigned at the node level.

To enable the use of the policy sets and bindings, you must also make the certificates and keys available to Integration Bus. You make the certificates and keys available by adding the keys to the keystore and configuring the truststore. The integration node is configured to use just one keystore for all such security artifacts.



## Assigning policy sets and bindings in the BAR file editor

- Select the message flow in the **Manage** tab to show the **Consumer Policy Set** and **Provider Policy Set** properties
- Click the **Edit** button to select a previously defined Policy Set and Policy Set Binding
- Can also set on a SOAP Input, SOAP Request, and SOAP AsyncRequest node for more control



© Copyright IBM Corporation 2016

Figure 7-45. Assigning policy sets and bindings in the BAR file editor

WM676/ZM6761.0

### Notes:

You can assign policy sets and policy set bindings on the message flow or on the SOAP nodes by using the Integration Toolkit BAR file editor.

## Unit summary

Having completed this unit, you should be able to:

- Access a message flow as a web service
- Request a web service from within a message flow
- Describe the functions of HTTP and SOAP nodes
- Generate WSDL files from message sets
- Describe the SOAP message tree
- Describe how WS-Addressing and WS-Security are implemented in IBM Integration Bus

© Copyright IBM Corporation 2016

Figure 7-46. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions (1 of 2)

1. True or False: Extra administrative actions are necessary to implement WS-Addressing.
  
2. True or False: Extra administrative actions are necessary to implement WS-Security.

© Copyright IBM Corporation 2016

Figure 7-47. Checkpoint questions (1 of 2)

WM676/ZM6761.0

### Notes:

Write your answers here:

1.

2.



## Checkpoint questions (2 of 2)

3. Name the main differences between HTTP and SOAP nodes.
4. Why must you explicitly import a WSDL file into a message set, rather than drag the file?
5. What information can be found in the `Root.SOAP.Context` folder of a message?

© Copyright IBM Corporation 2016

Figure 7-48. Checkpoint questions (2 of 2)

WM676/ZM6761.0

### Notes:

Write your answers here:

3.

4.

5.



## Checkpoint answers (1 of 2)

1. True or False: Extra administrative actions are necessary to implement WS-Addressing.

Answer: **False**. No extra administrative actions are necessary to implement WS-Addressing.

2. True or False: Extra administrative actions are necessary to implement WS-Security.

Answer: **True**.

© Copyright IBM Corporation 2016

---

Figure 7-49. Checkpoint answers (1 of 2)

WM676/ZM6761.0

### Notes:

## Checkpoint answers (2 of 2)

3. Name the main differences between HTTP and SOAP nodes.

Answer: SOAP nodes are configured through WSDL. They can be configured for WS-Addressing and WS-Security. They use the SOAP parser, which generates a specific tree.

4. Why must you explicitly import a WSDL file into a message set, rather than drag the file?

Answer: A deployable WSDL file is created that is synchronized with the `.mxsd` files (through annotations).

5. What information can be found in the `Root.SOAP.Context` folder of a message?

Answer: WSDL-derived information: Port, portType, Operation, operationType.

© Copyright IBM Corporation 2016

Figure 7-50. Checkpoint answers (2 of 2)

WM676/ZM6761.0

### Notes:

## Exercise 3



### Implementing web services

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 7-51. Exercise 3

WM676/ZM6761.0

### Notes:

In this exercise, you use the SOAP nodes to implement web services. You implement message flows to act as both a provider and a user of web services.



## Exercise objectives

After completing this exercise, you should be able to:

- Provide a message flow as a web service that uses SOAP/HTTP
- Start a SOAP/HTTP web service asynchronously
- Test a SOAP/HTTP message flow
- Implement WS-Addressing
- Use the TCP/IP Monitor to view the data as it passes through the SOAP/HTTP message transport

© Copyright IBM Corporation 2016

Figure 7-52. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.



# Unit 8. Developing integration solutions by using integration services

## What this unit is about

An integration service is a specialized application with a defined interface and structure that acts as a container for a web services solution. In this unit, you learn how to create and implement IBM Integration Bus integration services.

## What you should be able to do

After completing this unit, you should be able to:

- Design a service interface
- Implement web service operations as message flows
- Compare and contrast integration services with SOAP-enabled message flows
- Deploy integration services
- Test integration services
- Generate a JavaScript client API for an integration service

## How you will check your progress

Checkpoints

Lab Exercise:

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Design a service interface
- Implement web service operations as message flows
- Compare and contrast integration services with SOAP-enabled message flows
- Deploy integration services
- Test integration services
- Generate a JavaScript client API for an integration service

© Copyright IBM Corporation 2016

Figure 8-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- Services on IBM Integration Bus
- Defining integration services
- Deploying and testing integration services
- Generating a JavaScript Client API

© Copyright IBM Corporation 2016

Figure 8-2. Topics

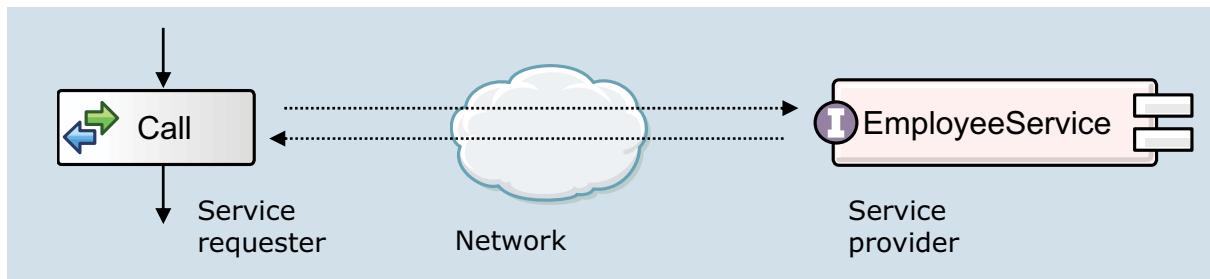
WM676/ZM6761.0

## Notes:



## 8.1. Services on IBM Integration Bus

## What is a service?



- A service is an application that is designed for machine-to-machine interaction over a network
  - Services describe their capabilities with a well-defined interface
  - Other systems interact with the service over a network
- A service exposes application functions in an interoperable format

© Copyright IBM Corporation 2016

Figure 8-3. What is a service?

WM676/ZM6761.0

### Notes:

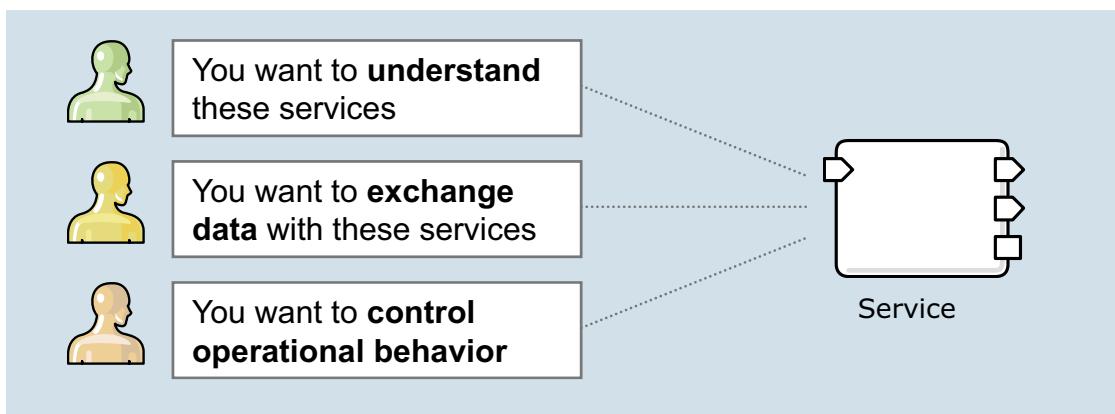
What is a service? A service is an application that is designed for machine-to-machine interaction over a network. Services describe their capabilities with a well-defined interface. The service hides its implementation details from the service requester. Other systems interact with the service over a network.

The purpose of a service is to expose application functions in an interoperable format.

A service provides application functions that are available through a standard interface. SOAP and REST web services are implementations of a service. A service-oriented architecture is an entire system that is implemented with services.

The definition of a service does not force you to use a specific interface format, such as WSDL. It does not specify a messaging format, such as SOAP.

## Integration challenges in a service-oriented world



- In a mixed implementation environment, each service provider and service consumer uses different syntax and format to describe services
- A service-oriented architecture must have a standard interface and policy to use, provide, govern, and catalog services

© Copyright IBM Corporation 2016

Figure 8-4. Integration challenges in a service-oriented world

WM676/ZM6761.0

### Notes:

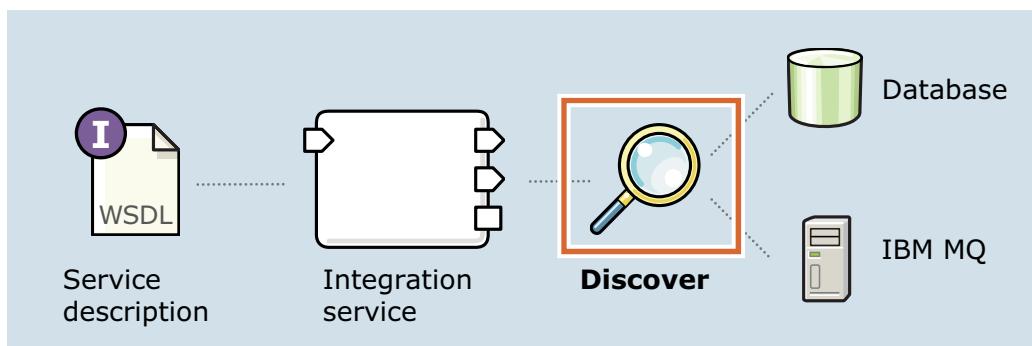
In a mixed implementation environment, each service provider and service consumer uses different syntax and format to describe services. For example, an IBM MQ service describes the data structure for its messages as a COBOL copybook. The service consumer defines a request message with XML schema.

A mixed implementation environment is also known as a heterogeneous environment. Many businesses have a mix of implementation types: IBM MQ applications, database applications, enterprise Java applications, and enterprise information systems. These systems define services in different ways.

The main goal of a service-oriented architecture is interoperability. However, if services do not use a standard syntax or format, service consumers and providers cannot talk to each other.

## Understanding services through service discovery

- **Service discovery** feature finds external client applications for use in the integration bus
  - Integration Toolkit interrogates the function declarations, data definitions, and communication objects from external IT systems
  - Developers select, elaborate, and refine the service definition
- A WSDL document defines the interface of the discovered service



© Copyright IBM Corporation 2016

Figure 8-5. Understanding services through service discovery

WM676/ZM6761.0

### Notes:

How do you gain an understanding of the services in your environment? The Integration Bus service discovery feature finds external client applications to use in the Integration Bus. The purpose of the service discovery feature is to survey and standardize server applications as SOAP web services. The WSDL document is the standard interface for these enterprise services.

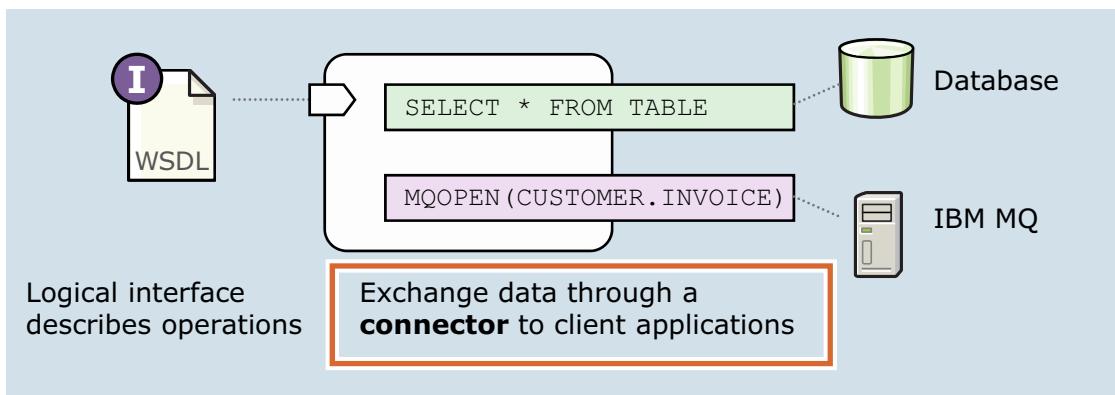
Two examples of external applications are database applications and IBM MQ applications.

The Integration Toolkit interrogates the function declarations, data definitions, and communication objects from external IT systems. Developers select, elaborate, and refine the service operation.

A WSDL document defines the interface of the discovered service.

## Exchanging data through connectors

- Integration services expose functions from IBM MQ and database services
  - Logical interface provides a standard, interoperable description of the service
- Exchange data from the logical service to the client applications through a connector
  - Configuration describes connection details and physical data exchange formats



© Copyright IBM Corporation 2016

Figure 8-6. Exchanging data through connectors

WM676/ZM6761.0

### Notes:

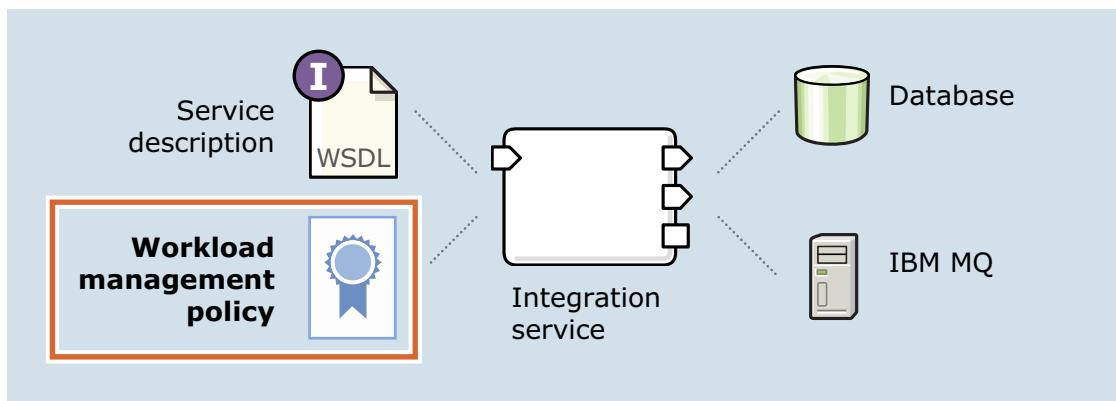
The integration service is a container for enterprise applications, in the form of a SOAP web service. The web service interface is a logical view to the IBM MQ or database applications. This interface provides a standard, interoperable description of the service.

To exchange data between the integration service and the enterprise applications, use a connector. The configuration for the connector describes connection details and physical data exchange formats.

The physical exchange format translates information from the integration service to the local data format of the enterprise application. For example, the integration service specifies SQL queries to retrieve information from a data source. The service uses IBM MQ API calls to manipulate data from an IBM MQ application.

## Control operational behavior through policies

- Attach workload management policies to control operational behavior
  - Measure and control the rate of messages in a message flow
  - Identify unresponsive flows and message thresholds
- Store workload management policies in the Integration Registry
  - Publish policies from the Integration Toolkit
  - Set and manage policies through the IBM Integration web interface, IBM Integration API, or command console



© Copyright IBM Corporation 2016

Figure 8-7. Control operational behavior through policies

WM676/ZM6761.0

### Notes:

You can apply workload management policies to control the operational behavior of deployed integration services. These policies can measure and control the rate of messages in a message flow. Policies can also identify unresponsive flows and message thresholds.

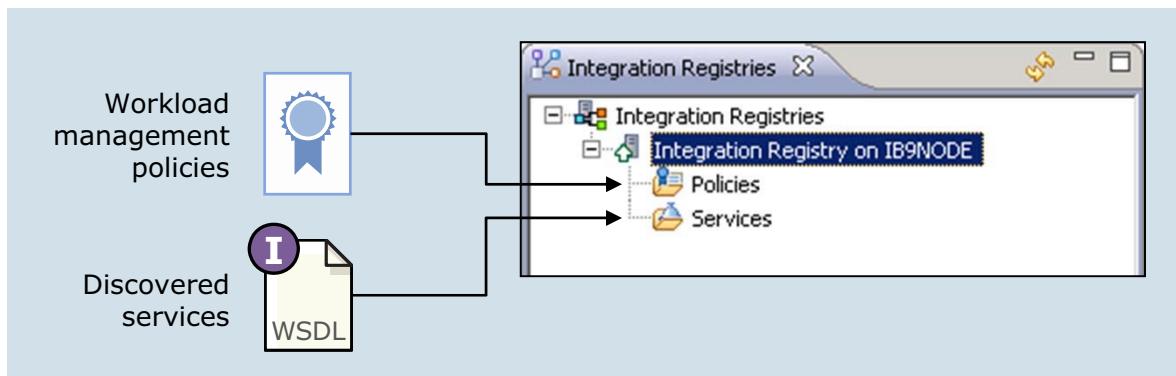
The Integration Registry maintains and stores a set of workload management policies in the integration server.

The Integration Registry is not designed to be a complete service registry solution. It does not replace the role of WebSphere Service Registry and Repository, or a Universal Description, Discovery, and Integration (UDDI) server. The Integration Registry collects WSDL documents of discovered services and WLM policies. It cannot catalog any other type of information.

You can set and manage policies through the IBM Integration web interface, Integration API, or command console. You can also create some policies, such as an MQEndpoint policy, in the Integration Toolkit.

## Integration Registry

- Publish service descriptions and workload management policies to the Integration Registry
  - Service providers can publish workload management policies and IBM MQ discovered services to facilitate reuse
  - Apply and modify policy settings at run time
  - The integration node hosts the Integration Registry



© Copyright IBM Corporation 2016

Figure 8-8. Integration Registry

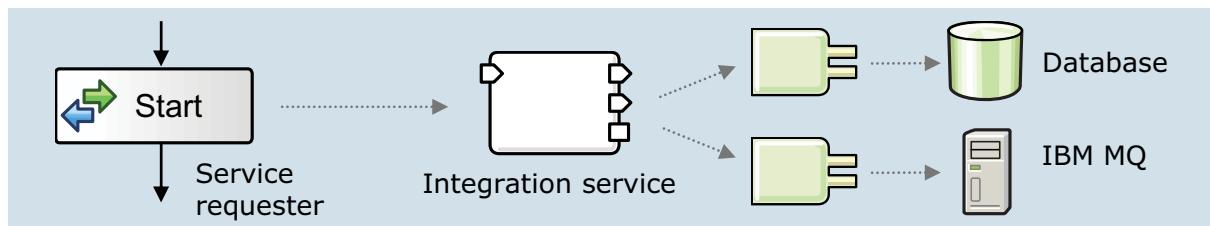
WM676/ZM6761.0

### Notes:

The Integration Registry stores service descriptions for discovered services and workload management policies to the Integration registry. Developers creating integration services can retrieve and incorporate discovered services in message flows. At run time, the administrator can apply workload management policies to services.

Each integration node hosts an Integration Registry.

## Integrating services with Integration Bus



- Integration services expose client applications in a standard, interoperable manner
  - Integration tools provide a common model for service consumers
  - Service discovery translates the common model to a provider-specific implementation
- The role of integration is relative
  - From the consumers' perspective, the integration service is the service
  - From the perspective of the database and IBM MQ system, the integration service is the consumer of its services
- Integration services are flexible
  - An integration service retrieves data from various sources, even a file

© Copyright IBM Corporation 2016

Figure 8-9. Integrating services with Integration Bus

WM676/ZM6761.0

### Notes:

With integration services, you can easily make enterprise applications available as standard, interoperable web services. Integration tools provide a common model for service consumers. Service discovery translates the common model to a provider-specific implementation.

The role of integration is a matter of perspective. From the point of view of the consumer, the integration service is the service. From the point of view of the enterprise application, the integration service is the consumer of its services.

Integration services are flexible. For example, an integration service retrieves data from various sources, even a file.

## **8.2. Developing integration services**

## What is an integration service?



- An integration service is a specialized application with a defined interface and structure that acts as a container for a web services solution
  - Contains message flows to implement the specified web service operations
  - Defines the interface with a WSDL document
- You can create an integration service in three ways:
  - Create an integration service from scratch
  - Start with an existing interface from a WSDL document
  - Start with an IBM Business Process Manager integration service

© Copyright IBM Corporation 2016

Figure 8-10. What is an integration service?

WM676/ZM6761.0

### Notes:

An integration service is a specialized application with a defined interface and structure that acts as a container for a web services solution. It contains message flows to implement the specified web service operations. It defines the interface with a WSDL document.

You can create an integration service in three ways:

- Create an integration service from scratch.
- Start with an existing interface from a WSDL document.
- Start with an IBM Business Process Manager integration service.

## Approaches to building web services

- You can build web services in three ways:
  - As an integration service
  - As a message flow with HTTP or SOAP nodes
  - Use REST APIs to expose integrations as a RESTful web service that HTTP clients can call
- Why use integration services?
  - It is a ready-built container for SOAP web services
  - It provides a clear mapping between web service operations and message flows
  - You can quickly build SOAP web services from database and IBM MQ applications
- When do you use a message flow with HTTP or SOAP nodes?
  - To build a web service that does not define a WSDL interface
  - To build a gateway for multiple services
  - To build services with different bindings, such as SOAP over JMS

© Copyright IBM Corporation 2016

Figure 8-11. Approaches to building web services

WM676/ZM6761.0

### Notes:

You can build web services in two ways: as an integration service, or as a message flow with HTTP or SOAP nodes.

Why should you use integration services?

First, it is a ready-built container for SOAP web services. It provides a clear mapping between web service operations and message flows. You can quickly build SOAP web services from database and WebSphere MQ applications.

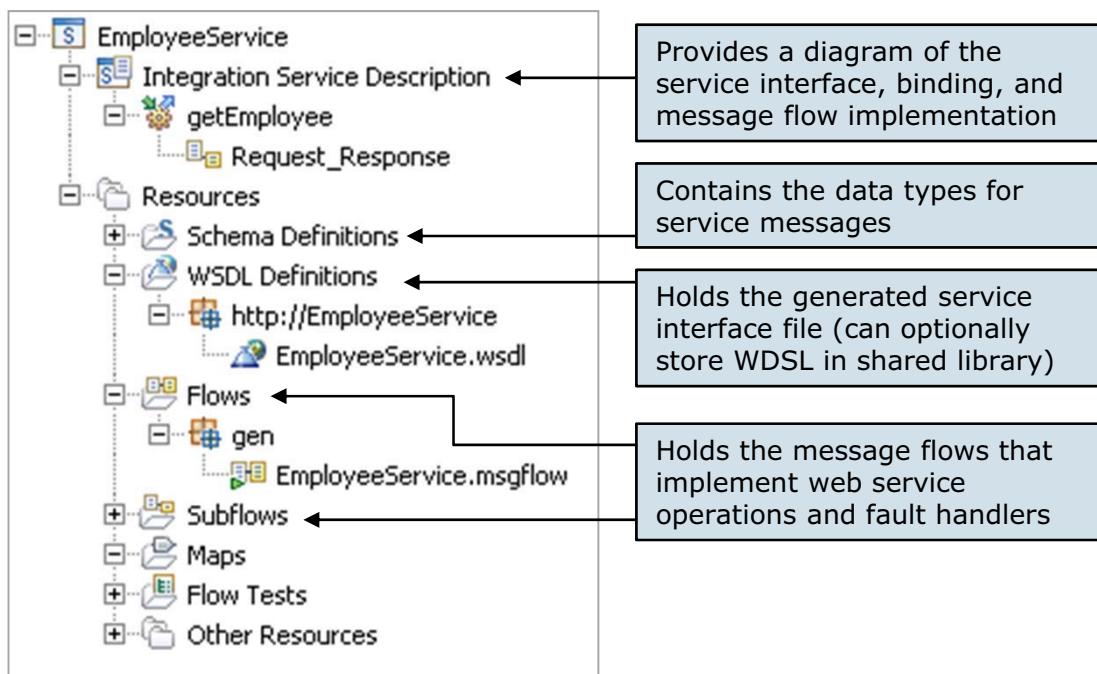
When should you use a message flow with HTTP or SOAP nodes instead of an integration service?

Use HTTP or SOAP nodes to build web services that do not define a WSDL interface. For example, RESTful services do not use the WSDL specification to define the interface or implementation details. In this case, use HTTP nodes to map REST resources to message flow features.

A common design pattern is a web service gateway: a single web service endpoint that represents more than one service on the server. To build this pattern, use a message flow.

Integration services support SOAP over HTTP connections only. To create web services with SOAP over JMS or IBM MQ bindings, use a message flow.

## Integration service project structure



© Copyright IBM Corporation 2016

Figure 8-12. Integration service project structure

WM676/ZM6761.0

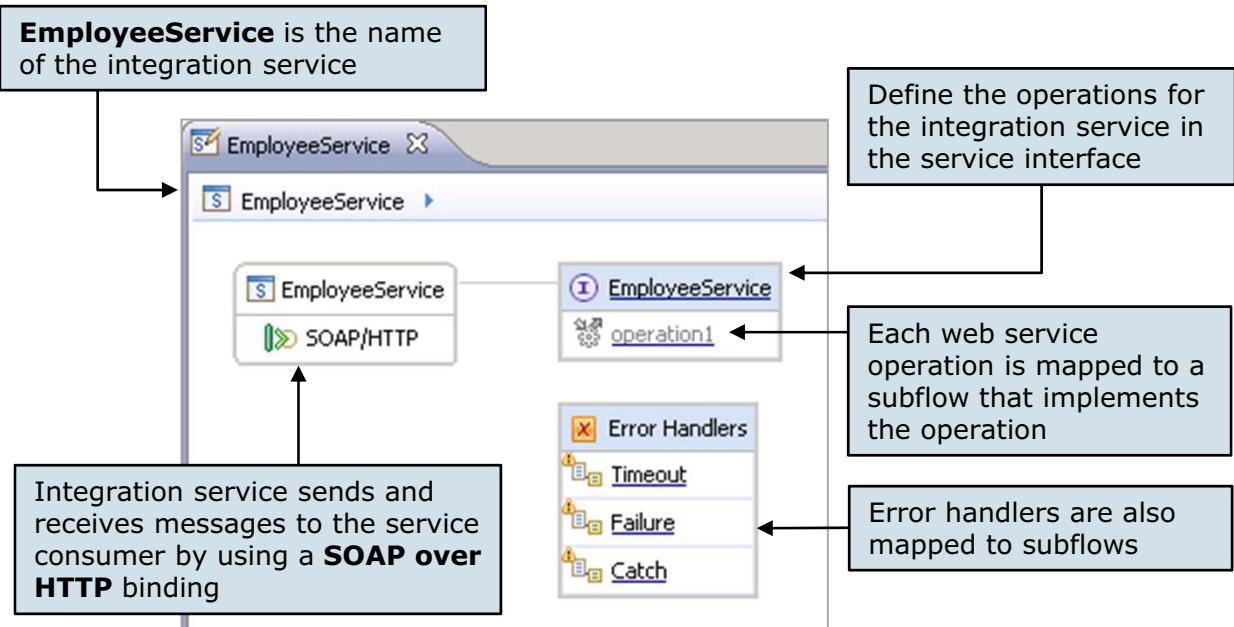
### Notes:

The main components of an integration service project are the Integration Service Description and the Resources.

The Integration Service Description provides a diagram of the service interface, binding, and message flow implementation.

The Resources include the schemas and the flows.

## Integration service editor



© Copyright IBM Corporation 2016

Figure 8-13. Integration Service editor

WM676/ZM6761.0

### Notes:

When you double-click the **Integration Service Description** link in an integration service project, the Integration Service editor opens. As shown in this example, the left side of the editor contains the integration service. It provides a SOAP over HTTP binding for its web service endpoint.

In this example, the EmployeeService service interface defines one operation: `operation1`.

A subflow defines the implementation for each web service operation.

Separate from the web service interfaces are three error handlers: **Timeout**, **Failure**, and **Catch**. You can customize the behavior for each of the three error handlers by editing the subflows for each handler.



## Service interface editor

**Interface**

Name	EmployeeService
Namespace	http://EmployeeService

**Operations**

Message Type	Name	Type
getEmployee	employee	EmployeeType
getEmployeeResponse	employee	EmployeeType
NoSuchEmployee	NoSuchEmployee	string

- View the name and namespace of the service interface
- Create request/reply and one-way web service operations
- Map XML schema data types to web service input, output, and fault messages

© Copyright IBM Corporation 2016

Figure 8-14. Service Interface editor

WM676/ZM6761.0

### Notes:

With the Service Interface editor, you can create web service operations for the integration service.

You can create a two-way request/reply operation with an input and output message. In this example, the **getEmployee** operation accepts an input message that is named **getEmployee**. The web service operation returns the result in a **getEmployeeResponse** message.

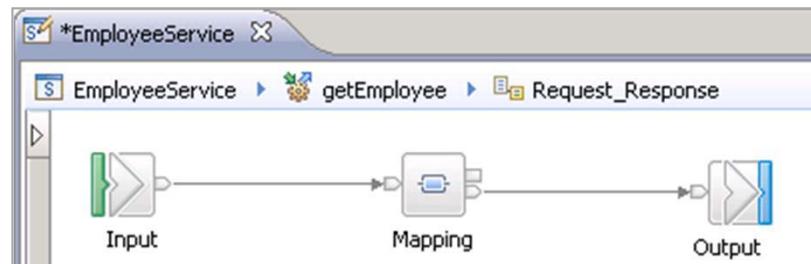
You can also create one-way web service operations. This type of operation has an input message only.

For both types of operations, you can specify one or more fault messages. Web service fault messages represent exception conditions that occurred in the web service execution.

You can also review the name and namespace for the integration service. However, you cannot edit those properties in this editor.

## Implementing service operation with flows

- The integration service generates a subflow for each operation in the service interface
- Input node maps the WSDL input message into the message assembly
- You define one or more processing nodes to process the web service request
- Output node copies the result of the message flow into the WSDL output message
  - One-way operations do not have an Output node



© Copyright IBM Corporation 2016

Figure 8-15. Implementing service operation with flows

WM676/ZM6761.0

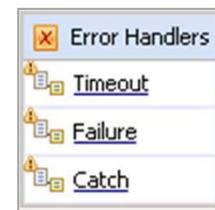
### Notes:

To implement an operation in the service interface, edit the message flow corresponding to the operation. The integration service generates a subflow for each operation in the service interface. The Integration Service editor also creates an input node and an output node, corresponding to the input and output messages for the service operation. Complete the implementation by adding one or more processing nodes in the subflow.

One-way operations do not return an output message to the client; the subflow for one-way operations does not have an output node.

## Handling errors with flows

- Web service operations can return error conditions with fault messages
  - In a request/reply operation, define a web service fault message in the service interface
  - In the web service operation message flow, return the fault message with an output node
  
- There are three default error handlers for all operations that the service interface defines:
  - If the HTTP connection exceeds the maximum connection time, integration service calls the **Timeout** error handler
  - If a message flow triggers a failure condition, integration service calls the **Failure** error handler
  - If a node generates an exception that is not handled within the message flow, integration service calls the **Catch** error handler



© Copyright IBM Corporation 2016

Figure 8-16. Handling errors with flows

WM676/ZM6761.0

### Notes:

Service operations can return error conditions with fault messages.

In a request/reply operation, define an integration service fault message in the service interface.

In the integration service operation message flow, return the fault message with an output node.

All operations have three default error handlers that are defined in the service interface:

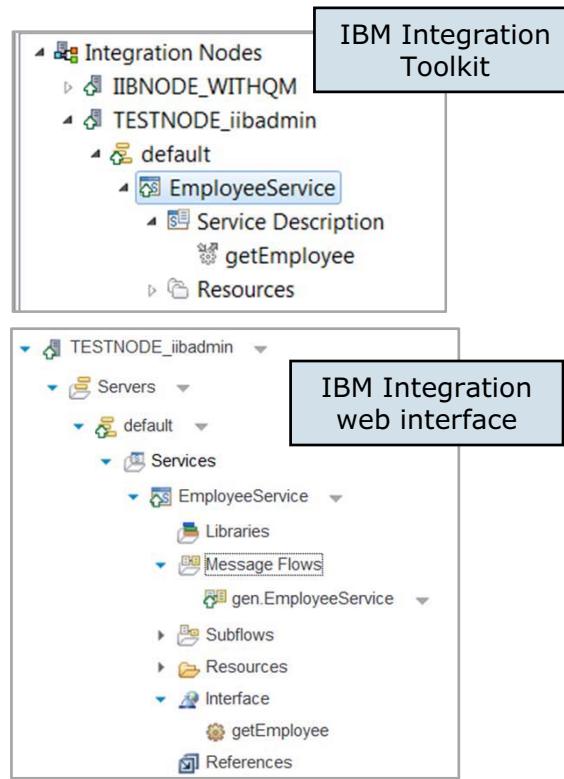
- If the HTTP connection exceeds the maximum connection time, the integration server calls the **Timeout** error handler.
- If a message flow triggers a failure condition, the integration server calls the **Failure** error handler.
- If a node generates an exception that is not handled within the message flow, the integration server calls the **Catch** error handler.

## **8.3. Deploying and testing integration services**



## Deploy the integration service

- Deploy integration services in the same manner as message flows
  1. Build a BAR file with the integration service resources
  2. Deploy the BAR to an Integration Bus integration server
- In the Integration Toolkit, view the service description, operations, and subflows in the **Integration Nodes** view
- In the Integration web interface:
  - View the service description, operations, and subflows under the integration server **Services** folder
  - Save the service interface files (WSDL and schemas)



© Copyright IBM Corporation 2016

Figure 8-17. Deploy the integration service

WM676/ZM6761.0

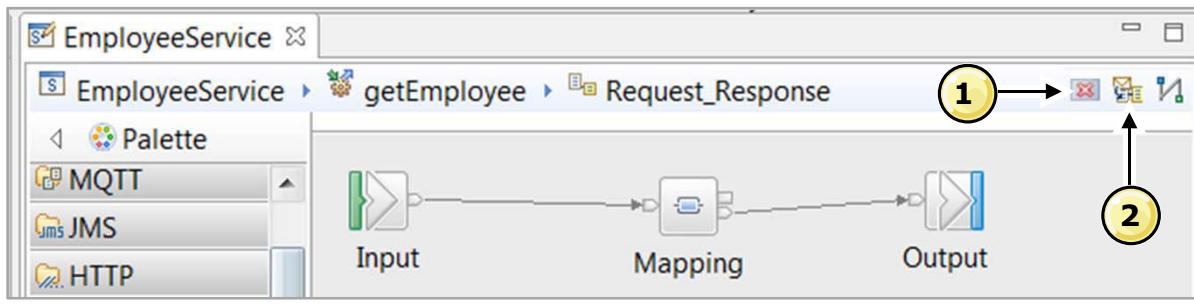
### Notes:

To deploy an integration service, follow the same steps as deploying a message flow to the integration server. Start by building a BAR file, with the integration service resources. Deploy the BAR file to an Integration Bus integration server.

In the **Integration Nodes** view, you can view the service description, operation, and their corresponding subflows.

## Testing the integration service (1 of 2)

- To test an integration service, send a test message with the Integration Toolkit Flow exerciser
- Edit the web service input message in the message section



© Copyright IBM Corporation 2016

Figure 8-18. Testing the integration service (1 of 2)

WM676/ZM6761.0

### Notes:

You can use the Integration Toolkit Flow exerciser to test the subflows in an integration service.

The Flow exerciser deploys the integration service to the selected integration server.



## Testing the integration service (2 of 2)

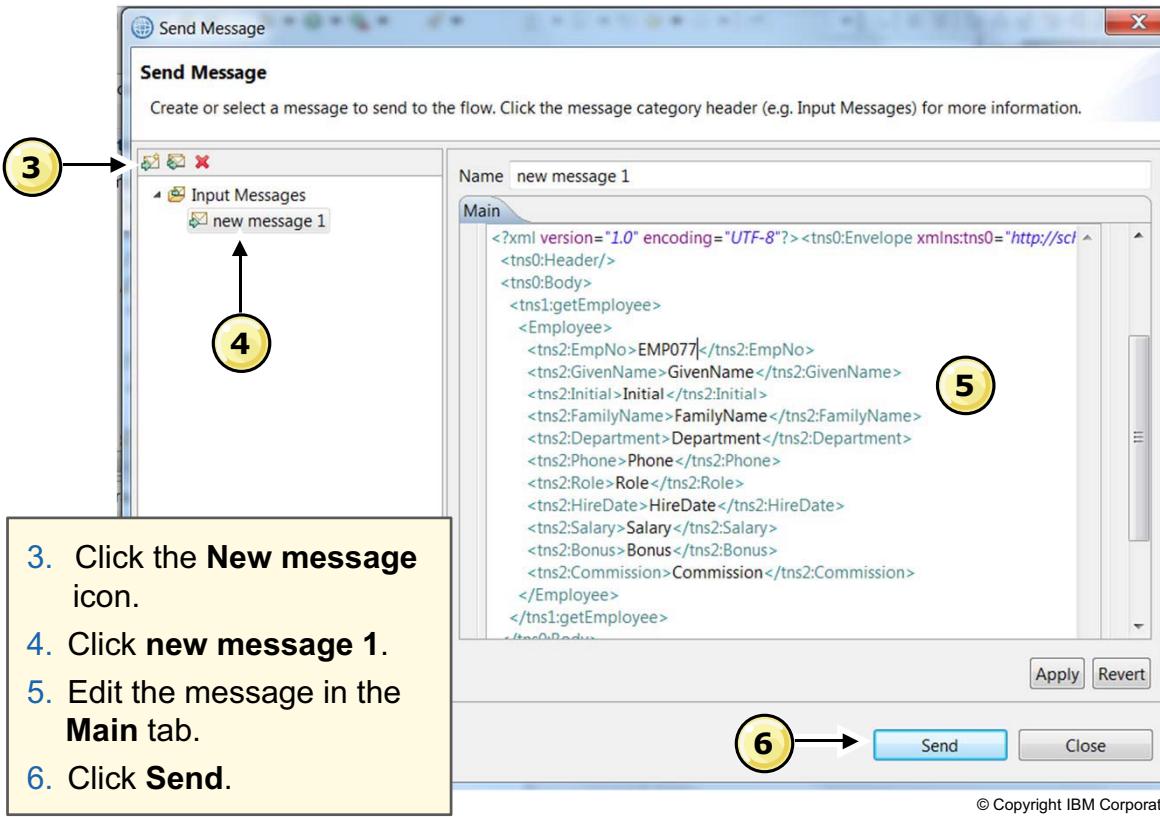


Figure 8-19. Testing the integration service (2 of 2)

WM676/ZM6761.0

### Notes:

In the Flow exerciser, you can edit the input message as an XML structure or read an input message from a file.

## Viewing message flow test results

- Confirm that the integration service completed successfully in the **Progress Information** view
- Examine the message in the Flow Exerciser

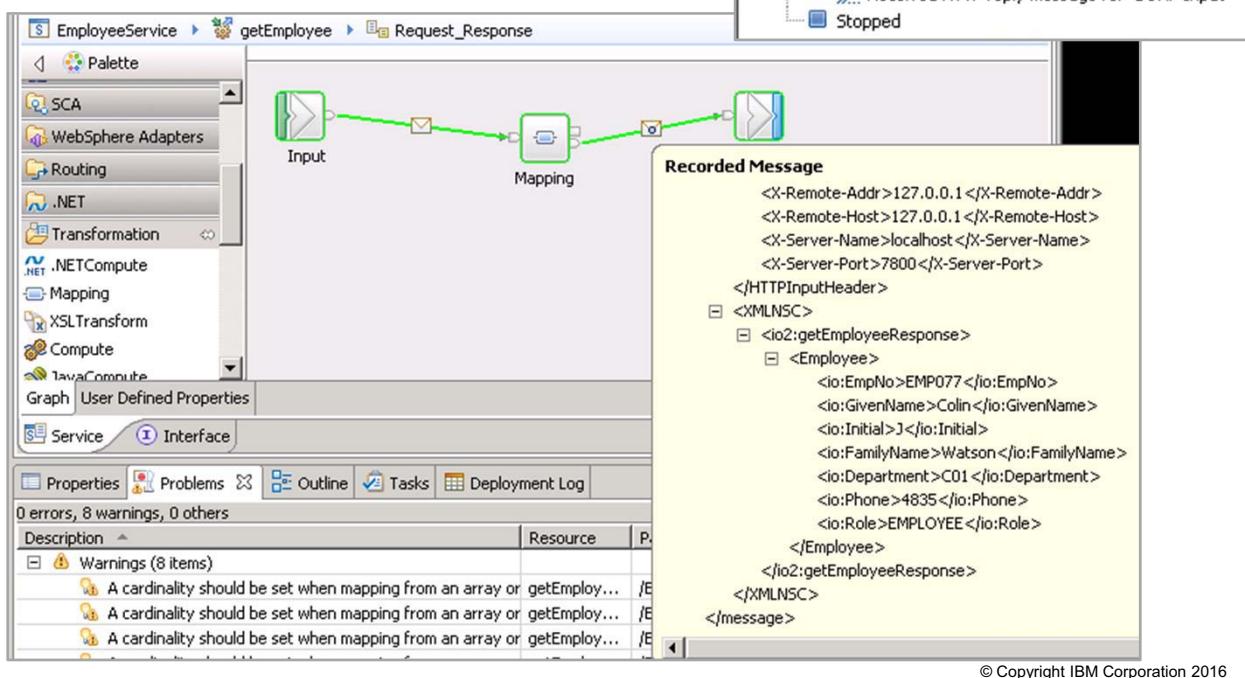


Figure 8-20. Viewing message flow test results

WM676/ZM6761.0

### Notes:

In the **Progress Information** view, confirm that the integration service completed successfully.



## 8.4. Generating a JavaScript client API



## JavaScript client API

- Generate a JavaScript client API from an existing integration service to provide operation functions that a JavaScript developer can call from an application that is running in a JavaScript environment
  - Generates JavaScript client API code from the definition of the existing integration service interface
  - Adds JSON/HTTP binding to the integration service so that it can process JSON messages that are sent from the JavaScript client API
  - Generates a web page that describes the JavaScript client API

© Copyright IBM Corporation 2016

Figure 8-21. JavaScript client API

WM676/ZM6761.0

### Notes:

You can generate JavaScript client API code from the definition of an existing integration service interface. You can use this code in JavaScript applications to call the integration service operations without configuring the underlying communication mechanism. A JavaScript method is created for each integration service operation.

A JSON/HTTP binding is added to the integration service so that the integration service can process JSON (JavaScript Object Notation) messages that are sent from the JavaScript client API. The JSON/HTTP binding is intended for use only by the JavaScript client API.

A web page that describes the JavaScript client API is generated. From the web page, the JavaScript developer can download or reference the JavaScript client API code and copy sample code directly into JavaScript client applications. The web page is accessed from the integration service URL.



## JavaScript API client restrictions

- Supported only from clients that either Node.js or a Google Chrome web browser initiates
- If you are deploying a web browser-based JavaScript application, then the Integration Bus HTTP proxy servlet must be deployed on the same web server as the JavaScript application
- The SOAP/HTTP binding is not required for the integration service to be called by the JavaScript client API
  - If the SOAP/HTTP binding is removed from an integration service, then the root URL for the integration service is not available and the JavaScript client API code and associated web page are not accessible

© Copyright IBM Corporation 2016

Figure 8-22. JavaScript API client restrictions

WM676/ZM6761.0

### Notes:

The figure lists the JavaScript API client restrictions.

## Work flow for Browser client

Integration Bus developer

Create and implement an integration service

Generate JavaScript client API for the integration service

Deploy and test the integration service

Configure Integration Bus proxy servlet and deploy to a servlet container

Browser JavaScript client developer

Browse the JavaScript client API page at the service endpoint

Write JavaScript client web page based on the example at service endpoint

Include the JavaScript client page in a deployable WAR file

Deploy WAR file to servlet container where the proxy servlet is located

Access the JavaScript client page from a browser

*Service endpoint*

© Copyright IBM Corporation 2016

Figure 8-23. Work flow for Browser client

WM676/ZM6761.0

### Notes:

This figure summarizes the work flow for a JavaScript API Browser client.

## Work flow for Node.js client

Integration Bus developer

Create and implement an integration service

Generate JavaScript client API for the integration service

Deploy and test the integration service

Node.js JavaScript client developer

Browse the JavaScript client API page at the service endpoint

Download the JavaScript API file for the service from the page

Ensure that the node package was downloaded

Write a JavaScript client based on the example at the service endpoint

Run the JavaScript client against the service

Service endpoint

© Copyright IBM Corporation 2016

Figure 8-24. Work flow for Node.js client

WM676/ZM6761.0

### Notes:

This figure summarizes the work flow for a Node.js client.

**Node.js** is an open source, cross-platform runtime environment for developing server-side web applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, Linux, FreeBSD, NonStop, IBM AIX, IBM System z and IBM i.

You can provide JavaScript developers with the integration service URL so that they can download or reference the JavaScript client API files and copy the sample code.



## Generate JavaScript client API for an integration service

1. In the IBM Integration Toolkit, open the integration service in the Integration Service editor.
2. Click the **Service** tab to display the SOAP/HTTP binding.
3. Above the SOAP/HTTP binding, right-click the integration service name and the click **Generate > JavaScript Client API**.

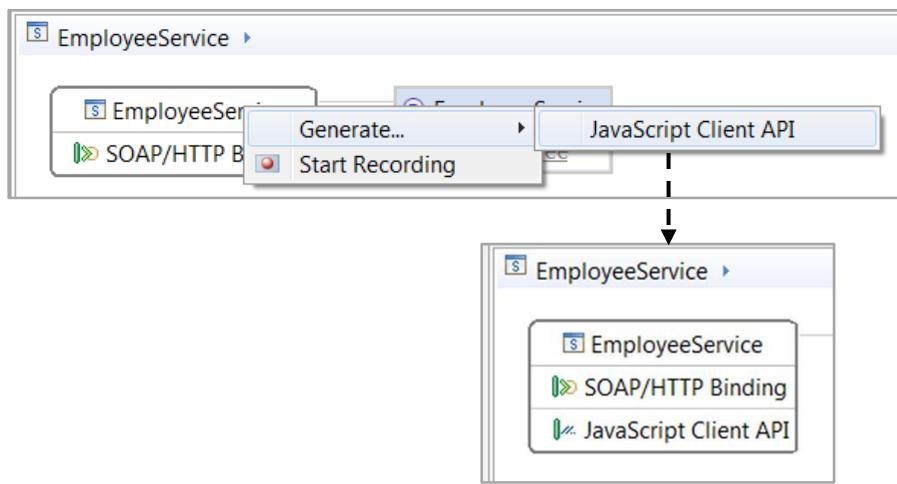


Figure 8-25. Generate JavaScript client API for an integration service

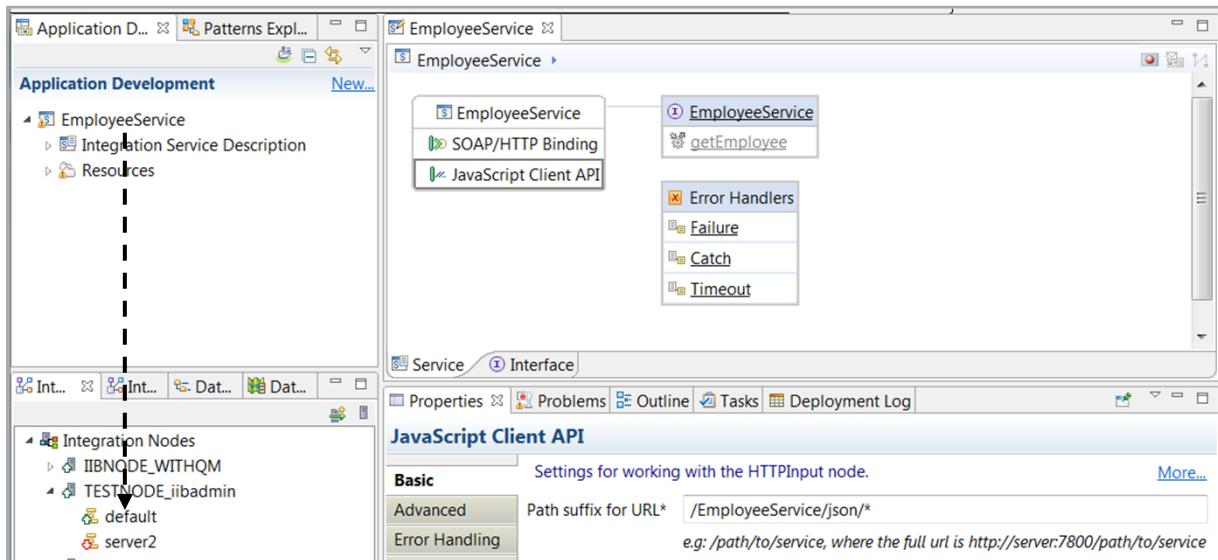
WM676/ZM6761.0

### Notes:

You generate a JavaScript client API for an integration service in the Integration Toolkit.



## Deploy the integration service



© Copyright IBM Corporation 2016

Figure 8-26. Deploy the integration service

WM676/ZM6761.0

### Notes:

After you generate the JavaScript client API, deploy the integration service so that you can access the Service URL.

**WebSphere Education**

**View the generated resources**

The screenshot shows the IBM Integration Bus V10 Development II interface. On the left, there's a tree view of 'Integration Nodes' containing 'IIBNODE\_WITHQM', 'TESTNODE\_jibadmin' (which has 'EmployeeService' selected), and 'TESTNODE\_wattss'. On the right, the 'Properties' tab is active, showing details for 'EmployeeService'. A callout box points to the 'Service URL' field with the instruction: 'Get the Service URL value from the Integration Service properties and paste into browser window'. Below this, a browser window is open at the URL <http://9.76.5.15:7800/EmployeeService>. The browser title bar says 'Integration Service: EmployeeService'. The page content includes the heading 'Integration Service: EmployeeService' and a note: 'This integration service can be invoked using: SOAP / HTTP, JavaScript Client API'. A callout box points to the 'Click JavaScript Client API' link.

Property	Value
Info	
Deployment Time	Wed Nov 04 14:35:58 IST 2015
Full Name	getEmployee
Last Modified	<a href="http://9.76.5.15:7800/EmployeeService?wsdl">http://9.76.5.15:7800/EmployeeService?wsdl</a>
Operations	
Service Query URL	
Service URL	<a href="http://9.76.5.15:7800/EmployeeService">http://9.76.5.15:7800/EmployeeService</a>

Figure 8-27. View the generated resources

WM676/ZM6761.0

### Notes:

To view the generated resources, copy the Service URL from the service properties and then paste it in a browser. In the web page, click the JavaScript Client API link to view open the resources page.



## Generated resources

### Invoke using JavaScript Client API

#### Instructions

1. Set up the JavaScript client environment
2. Install the npm dojo package using 'npm install dojo' (only if you are developing in a Node.js environment)
3. Download the EmployeeService.js file
4. Write a JavaScript application which calls the integration service JavaScript methods

#### File

[EmployeeService.js](#) - JavaScript method(s) for this integration service

**Method:** IBMIntegration.EmployeeService.getEmployee()

#### Description

*None.*

#### Input

`Employee : EmployeeType`

#### Output

`Employee : EmployeeType`

#### Coding Example

```
/* Uncomment these lines if you are developing in a Node.js environment.

require("http");
require("./EmployeeService");

IBMIntegration.EmployeeService.IBMContext.hostname = "9.76.5.15";
IBMIntegration.EmployeeService.IBMContext.port      = 7800;

*/
```

For node.js, create a file in the **c:\nodejs** folder that is called **EmployeeService.js** and copy all the contents of the generated JavaScript file from the web browser into the file

Resources include coding example

© Copyright IBM Corporation 2016

Figure 8-28. Generated resources

WM676/ZM6761.0

## Notes:

This figure is an example of the generated service for the EmployeeService. The resources include sample code.



## Unit summary

Having completed this unit, you should be able to:

- Design a service interface
- Implement web service operations as message flows
- Compare and contrast integration services with SOAP-enabled message flows
- Deploy integration services
- Test integration services
- Generate a JavaScript client API for an integration service

© Copyright IBM Corporation 2016

Figure 8-29. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or false: Integration services are best suited to create web services.
2. True or false: Integration services are best suited to create SOAP web services.

© Copyright IBM Corporation 2016

Figure 8-30. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

1.

2.



## Checkpoint answers

1. True or false: Integration services are best suited to create web services.  
Answer: **False.** Integration services are the simplest way to build a service for Integration Bus. However, they do not support all types of web services.
  
2. True or false: Integration services are best suited to create SOAP web services.  
Answer: **False.** Integration services do not support the JMS transport for messages. Therefore, SOAP over JMS is not supported.

© Copyright IBM Corporation 2016

---

Figure 8-31. Checkpoint answers

WM676/ZM6761.0

### Notes:

## Exercise 4



Creating an integration service

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 8-32. Exercise 4

WM676/ZM6761.0

### Notes:



## Exercise objectives

After completing this exercise, you should be able to:

- Create a container for an integration service
- Design the operations, bindings, and messages for a service interface in a graphical editor
- Implement a service operation with a database SQL call
- Deploy and test an integration service by using the IBM Integration Toolkit Flow exerciser
- Export the WSDL document that describes the service by using the IBM Integration web user interface

© Copyright IBM Corporation 2016

---

Figure 8-33. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.

# Unit 9. Connecting a database by using a discovered service

## What this unit is about

An IBM Integration Bus database service defines the interaction between Integration Bus and a database, and can be used with a message flow node to automatically configure connectivity to that database. In this unit, you learn how to define and implement a database service.

## What you should be able to do

After completing this unit, you should be able to:

- Create a database service
- Use the database service to populate node properties

## How you will check your progress

- Checkpoints
- Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus V10



## Unit objectives

After completing this unit, you should be able to:

- Create a database service
- Use the database service to populate node properties

© Copyright IBM Corporation 2016

---

Figure 9-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Service discovery

- Builds a description of service interfaces
  - Service descriptions to use the service, provide the service, govern the service, or catalog the service
  - In a heterogeneous environment, each service provider can have different syntax and format for describing the service
- Provides a common syntax and format for describing all services
  - Connectors are required to translate the service provider description to a common model
  - Discovery tools serialize the common model to standard syntax

© Copyright IBM Corporation 2016

Figure 9-2. Service discovery

WM676/ZM6761.0

### Notes:

The Integration Toolkit Service Discovery editor can build a description of service interfaces for a database service and an IBM MQ service.

- Use a database service to make database operations accessible both within a message flow, and to external applications that call a message flow.
- Use an IBM MQ service to discover artifacts from IBM MQ queue managers. You can then use these artifacts in a message flow. You can use an IBM MQ service to configure the properties of an IBM MQ node. You can reuse an IBM MQ service across multiple nodes, which simplifies the IBM MQ connectivity within your message flows.

The Service Discovery editor has six pages that guide you through the creation of a database service. When the Service Discovery editor is started, the editor focus is on the **Description** pane. When an already discovered service is started, the editor focus is on the **Service Interface** pane. You can use the **Checklist** in the editor to move between panes.

## Database services

- Database services define interaction between Integration Bus and databases in a service-oriented fashion
  - Each database service can support operations only on a single database
  - Support is restricted to DB2 databases
- Service description is captured in a WSDL document
  - *Port type* defines service interface
  - *Database binding* captures database connector details
  - *Service operations* map onto database operations
- Message flows can reference database services in a Compute node

© Copyright IBM Corporation 2016

Figure 9-3. Database services

WM676/ZM6761.0

### Notes:

You can create a database service in the Database Service editor by combining metadata that is discovered from the database, with information supplied by the flow developer about the required database interaction. The Database Service editor uses a database definition file (.dbm file) that contains information about a connection to a database, and typically information about the database contents such as the database schemas, tables, and stored procedures.

The service description generates a WSDL file that contains the following information:

- Operation-oriented interface definitions about the database interaction
- A database connection binding that captures the semantics of the operations on the database

XML schema files describe the inputs and outputs of the database operations.

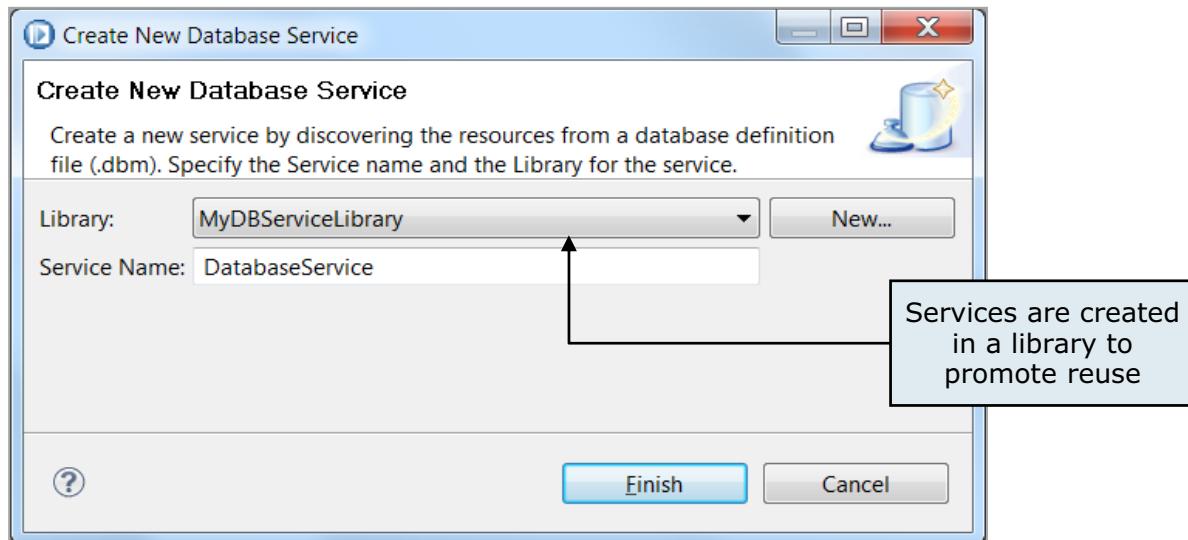
A database service has the following limitations:

- It can be used with a Compute node only.
- It can interact with a DB2 database only.
- Each database service can support operations on one table in the database.



## Starting the Database Service editor

- Click **File > New > Database Service** or
- Click **Start by discovering a service** from the **Quick Start** menu



© Copyright IBM Corporation 2016

Figure 9-4. Starting the Database Service editor

WM676/ZM6761.0

### Notes:

You can start the Database Service editor by using one of the following methods:

- Click **File > New > Database Service**.
- In the Application Development view, click **New > Database Service**.
- Right-click the white space of the **Application Development** view, and then click **New > Database Service**.
- Click **Start by discovering a service** from the **Quick Start** menu.

Specify names for the library and the service, and click **Finish**.

**WebSphere Education**

**Database Service editor: Description**

A Database Service defines a business service that is available in a database, such as tables, views, and the logical business operations they represent. You interact (SELECT, INSERT, UPDATE, DELETE) to create a Database Service.

**Integration Service**

Legend  
Logical Physical

When you save the Database Service, the following artifacts are generated:

- <DatabaseService>.wsdl: This file describes the logical service interface database.
- <DatabaseService>.xsd: This file describes the inputs and outputs of the service.
- <DatabaseService>.service: This file captures other information that is used by the ESQL module.

Use the Database Service in a message flow by dragging the Database Service icon from the palette and dropping it into the message flow. Then configure connectivity to a database. In your ESQL module, call the Database Service.

Click [here](#) for additional information on Database Services.

© Copyright IBM Corporation 2016

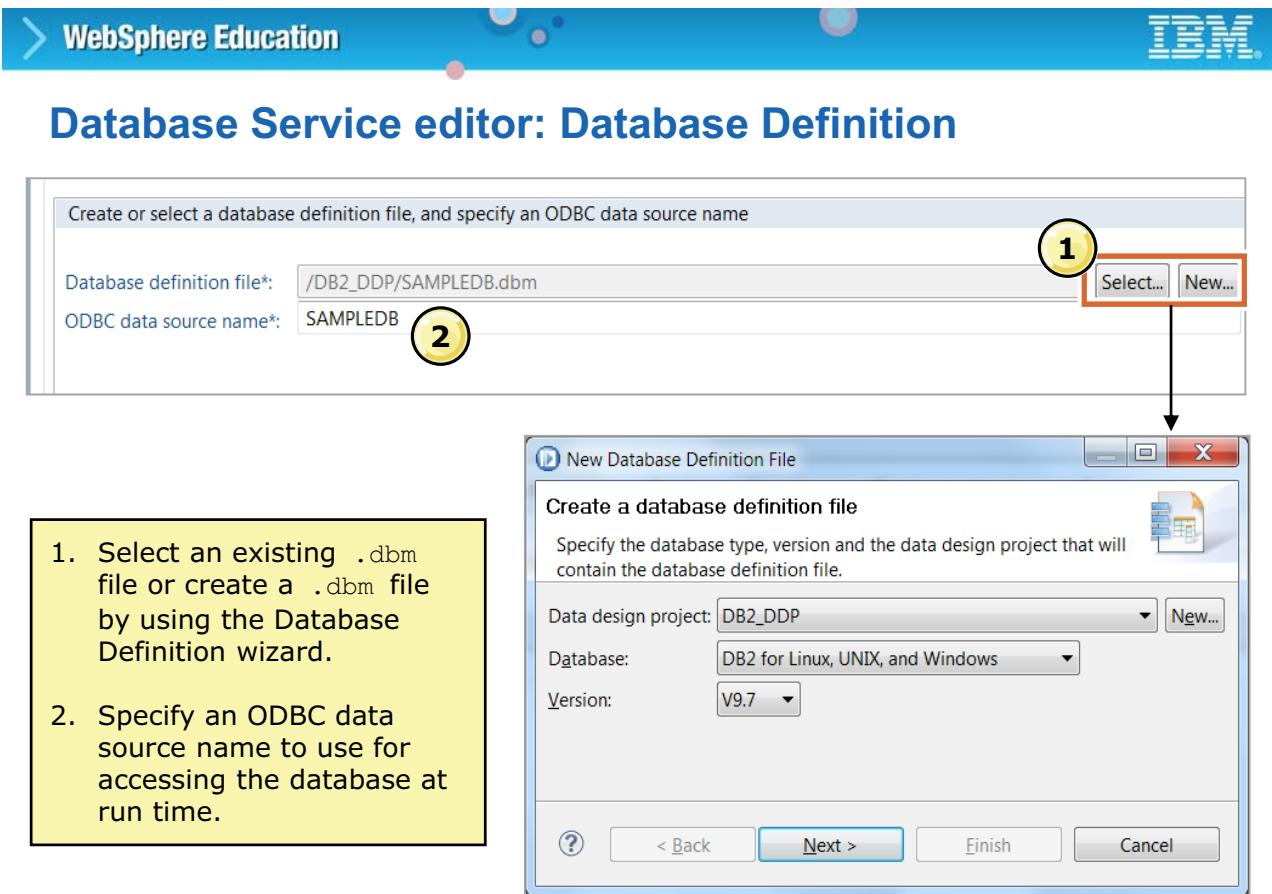
Figure 9-5. Database Service editor: Description

WM676/ZM6761.0

## Notes:

The Database Service editor contains six pages. You can the option in the **Checklist** in the Database Service editor to go directly to any page.

The Database Service editor opens on the **Description** page. The **Description** page contains the steps for defining a database service and a description of the artifacts that the Database Service editor generates.



© Copyright IBM Corporation 2016

Figure 9-6. Database Service editor: Database Definition

WM676/ZM6761.0

## Notes:

On the **Database Definition** page, you discover the database definition by using one of the following methods:

- Specify the name of an existing Integration Bus database definition (.dbm) file to import.
- Click **New** to create a database definition file for the database.

You must also specify the ODBC data source name. The Database Definition step assumes that an ODBC data source is already created.



### Note

You must verify that the ODBC data source name that is automatically generated matches the runtime environment ODBC data source name for the database.

## Database Service editor: Select Resources

Select resources

Select the single table you wish to expose operations from

SAMPLEDB

- SAMPLEF
- CUSTOMER (highlighted with a red box)
- INVENTORY
- CREDITTB
- PRICETB
- ORDERTB
- ITEMTB
- PURCHASEORDER
- ITEMS
- EMPLOYEE
- CREDITSCORE

Preview of columns in selected table: CUSTOMER

CUSTOMERID
PREFIX
FIRSTNAME
MIDDLENAME
LASTNAME
TITLE
CUSTOMERTYPE
CUSTOMERNAME
OFFICEPHONENO
MOBILEPHONENO
FAXNO
EMAILADDRESS
BILLSTREETNO

Select a table for which you want to create the database services

Lists database and tables that the Integration Bus database file (.dbm) defines

Shows columns of the selected tables to help you identify the table to select for the database service

© Copyright IBM Corporation 2016

Figure 9-7. Database Service editor: Select Resources

WM676/ZM6761.0

### Notes:

On the **Select Resources** page, you select the table for which you want to provide operations.



You can select only one database table.

When you select a table, the table columns are shown in the preview pane to help you to determine the table to select.

## Database Service editor: Service Operations (1 of 3)

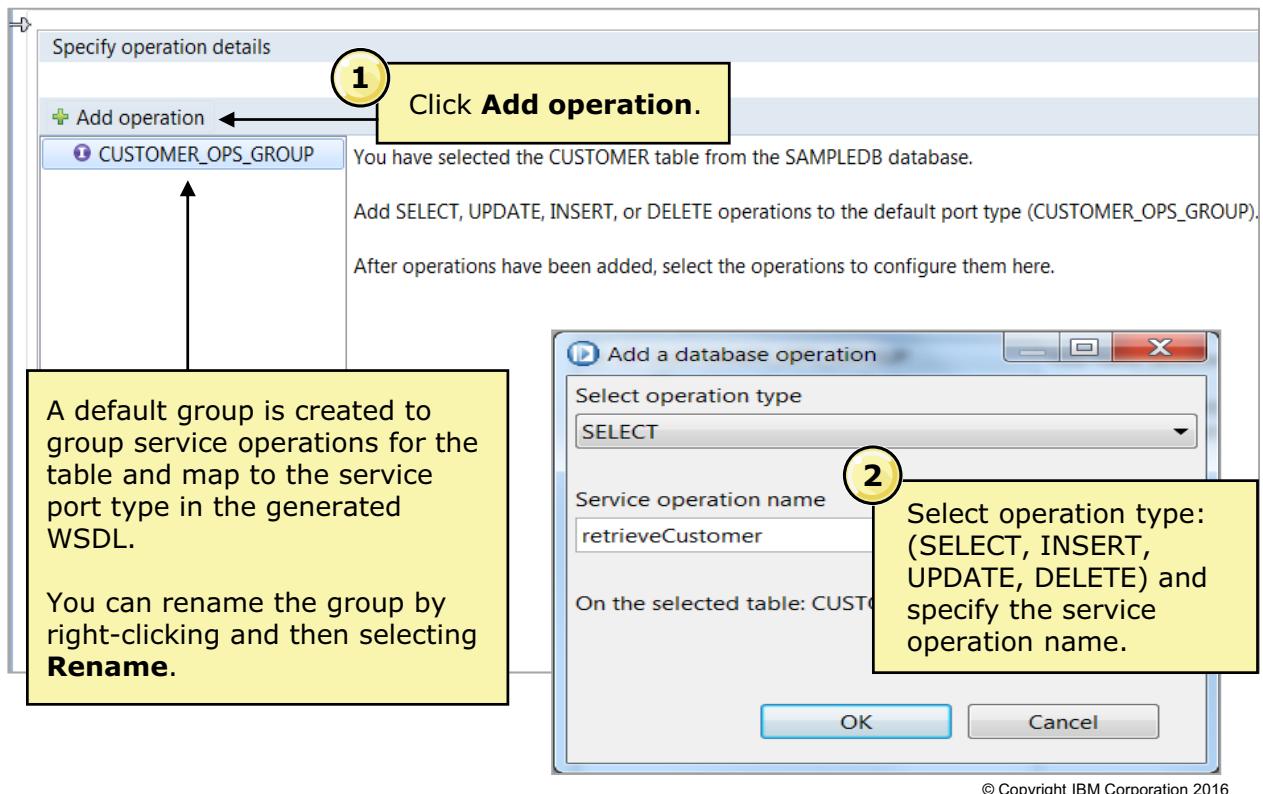


Figure 9-8. Database Service editor: Service Operations (1 of 3)

WM676/ZM6761.0

### Notes:

On the **Service Operations** page, add the required SELECT, UPDATE, INSERT, or DELETE service operations. You can define several service operations, but they can relate only to the one database table.

A default group is created to group the service operations for the table. By default, the group is named <table\_name>\_OPS\_GROUP. In the example, the group is named CUSTOMER\_OPS\_GROUP because the CUSTOMER table was selected on the **Select Resources** page. This group maps to the service port type in the WSDL that the Database Service editor generates. You can rename the group by right-clicking the group name and then clicking **Rename**.

To add an operation:

1. Click **Add operation**.
2. Select the operation type and then enter a descriptive service operation name.

## Database Service editor: Service Operations (2 of 3)

Specify operation details

**i** Operation Type: SELECT -- Selected columns from the table will be output parameters of the operation. In the Output Columns tab set the sort preferences for the columns. In the Conditions tab set which rows will be returned.

+ Add operation

CUSTOMER\_OPS\_GROUP

- retrieveCustomerByID
- retrieveCustomersByLastName**
- retrieveCustomersInToronto
- createCustomer
- updateCustomer
- deleteCustomer

**3** Select an operation to configure a corresponding database query.

**4** Select columns from the table and their sort order.

Selected columns are output parameters of the service operation.

	Sort Type	Sort Order
CUSTOMER.CUSTOMERID		
CUSTOMER.PREFIX	Ascending	2
CUSTOMER.FIRSTNAME	Ascending	1
CUSTOMER.MIDDLENAME		
CUSTOMER.LASTNAME		
CUSTOMER.TITLE		

© Copyright IBM Corporation 2016

Figure 9-9. Database Service editor: Service Operations (2 of 3)

WM676/ZM6761.0

### Notes:

The next steps on the **Service Operations** page are:

3. Select the operation.
4. Select the table columns that the service operation uses.



## Database Service editor: Service Operations (3 of 3)

Specified parameter name becomes the input parameter of the service operation

5 Build condition criteria by using one or more columns from the table.

The screenshot shows the Database Service editor interface. On the left, under 'Specify operation details', it says 'Operation Type: SELECT -- Selected columns from the table will be output parameters for the columns. In the Conditions tab set which rows will be returned'. Below this is a tree view of operations under 'CUSTOMER\_OPS\_GROUP' including 'retrieveCustomerByID', 'retrieveCustomersByLastName', etc. A yellow callout box labeled '5' points to the 'Conditions' tab in the 'Output Columns' section, where 'LASTNAME' is listed with an '=' operator and a value field containing ':lastname'. An arrow points from this field to the 'Value' field in the 'Expression Builder' dialog. The 'Expression Builder' dialog title is 'Expression Builder' and its sub-title is 'Input Value Options: Select the type of input options.' It shows 'Input parameter' selected with a value of ':lastname'. Other options like 'Numeric constant' and 'String constant' are also shown. A dropdown menu in the 'Value' field of the main editor is open, showing 'Build Expression...', 'Input Parameter', 'CUSTOMER.CUSTOMERID', 'CUSTOMER.PREFIX', and 'CUSTOMER.FIRSTNAME'. The bottom right of the dialog has 'Finish' and 'Cancel' buttons.

Figure 9-10. Database Service editor: Service Operations (3 of 3)

WM676/ZM6761.0

### Notes:

The next step is to define the operation conditions. For example, if the operation type is SELECT, specify the query condition to create a complete SELECT statement.

The value that you specify in the **Value** column becomes the input parameter of the service operation. When you click in the **Value** column, you have the option of selecting a column or starting the **Expression Builder**.

## Database query details

Specify operation details

**i** Operation Type: SELECT -- Selected columns from the table will be output parameters of the operation. In the preferences for the columns. In the Conditions tab set which rows will be returned.

+ Add operation

- CUSTOMER\_OPS\_GROUP
  - retrieveCustomerByID
  - retrieveCustomersByLastName
  - retrieveCustomersInToronto
  - createCustomer
  - updateCustomer
  - deleteCustomer

**SQL view** contains read-only view of the SQL that is generated based on selection of **Output Columns** and **Conditions**

By default, this view is collapsed

Expand, collapse, or resize any panes

```
SELECT CUSTOMERID, PREFIX, FIRSTNAME, MIDDLENAME, LASTNAME, TITLE,
       CUSTOMERTYPE, OFFICEPHONENO, MOBILEPHONENO
  FROM SAMPLE.CUSTOMER
 WHERE LASTNAME = :lastname
 ORDER BY LASTNAME ASC, FIRSTNAME ASC
```

CUSTOMER				
<input checked="" type="checkbox"/> CUSTOMERID	<input checked="" type="checkbox"/> PREFIX	<input checked="" type="checkbox"/> FIRSTNAME	<input checked="" type="checkbox"/> MIDDLENAME	<input checked="" type="checkbox"/> LASTNAME

Output Columns			Conditions	
Column	Operator	Value		
LASTNAME	=	:lastname		

SQL view

Table view

Column selection and conditions

© Copyright IBM Corporation 2016

Figure 9-11. Database query details

WM676/ZM6761.0

### Notes:

When you are defining a retrieve operation, it sends a SELECT statement to the database. In the Database Service editor **Service Operations** page, you can open an SQL view pane that contains a read-only of the SQL that results from your selections in the **Output Columns** and **Conditions**.

You cannot change the SELECT statement in the SQL view. If the statement is not correct, check the columns that are selected and the conditions.



## Database Service editor: Service Interface

Review the logical interfaces and operations created in this service

Interface	
Name	CUSTOMER_OPS_GROUP
Namespace	http://MyDBService

Service port type name is identical to the operations group name

Operations	
Operation	Message
retrieveCustomerByID	
<input type="button" value="Input"/>	<input type="button" value="retrieveCustomerByID"/>
<input type="button" value="Output"/>	<input type="button" value="retrieveCustomerByIDResponse"/>
retrieveCustomersByLastName	
<input type="button" value="Input"/>	<input type="button" value="retrieveCustomersByLastName"/>
<input type="button" value="Output"/>	<input type="button" value="retrieveCustomersByLastNameResponse"/>
retrieveCustomersInToronto	
<input type="button" value="Input"/>	<input type="button" value="retrieveCustomersInToronto"/>

Service Namespace is derived from service name

Click **Open file** to open the XSD schema file and review contents of request/response types for the operation

[Open file...](#)

[Open file...](#)

[Open file...](#)

[Open file...](#)

[Open file...](#)

- Iteratively build the service interface and review input and output types of the operations
- Go back to the **Service Operations** page, modify selections, and review the changes in **Service Interface**

© Copyright IBM Corporation 2016

Figure 9-12. Database Service editor: Service Interface

WM676/ZM6761.0

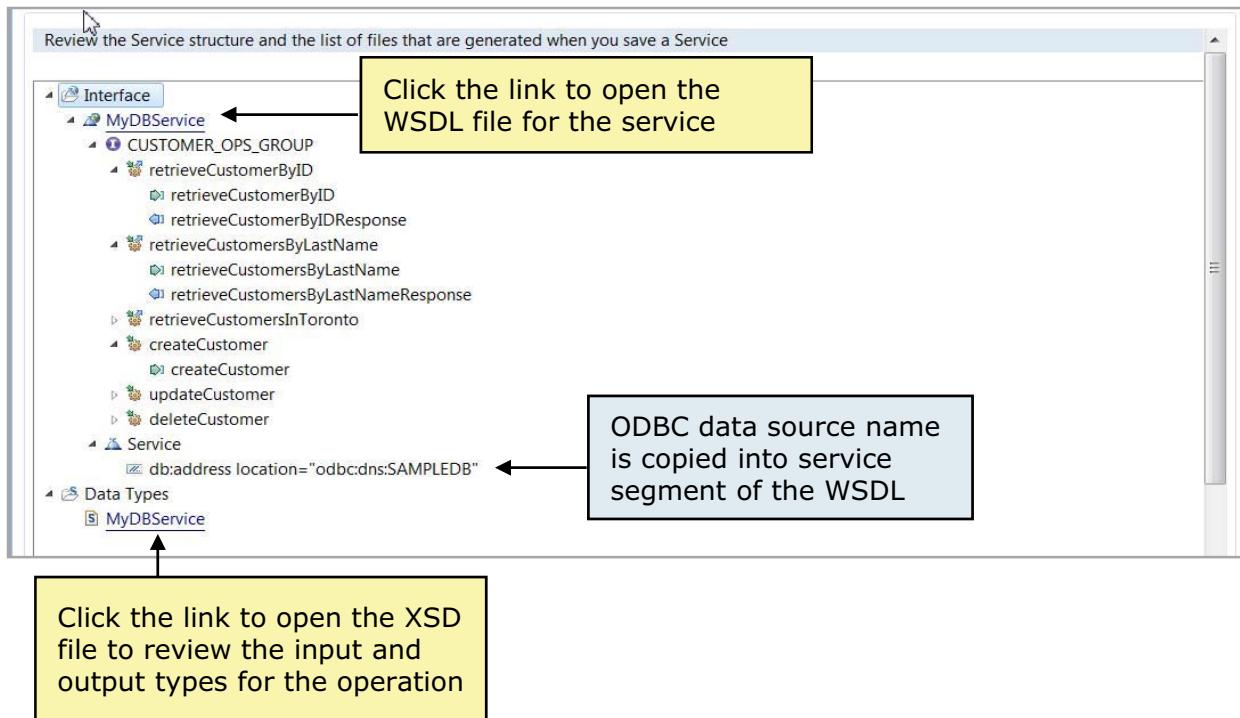
### Notes:

The next step is to define the service interface.

The **Interface** section contains the group name and the namespace. The service port type name in the WSDL is the same as the interface name. The namespace is derived from the service name.

The **Operations** section contains an entry for each operation that you create on the **Service Operations** page. For any interface, you can click the **Open file** link to open the XML schema for the operation. If the database service is not saved, clicking **Open file** prompts you to save the service file, which generates the WSDL and XSD files for service.

## Database Service editor: Summary



© Copyright IBM Corporation 2016

Figure 9-13. Database Service editor: Summary

WM676/ZM6761.0

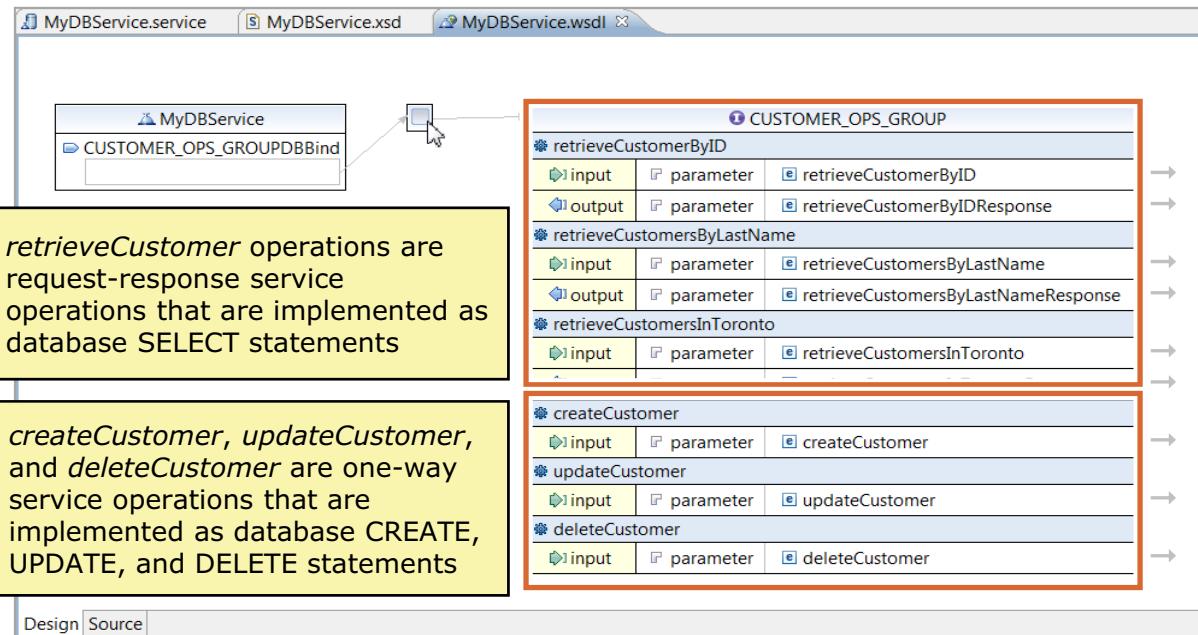
### Notes:

On the Database Service editor **Summary** page, you can review the service interface, which includes the operations and input and output message types.

You can click the service link to open the WSDL for the service. You can also click the link under the **Data Types** folder to open the XML schema file to review the input and output types for the operation.



## Generated WSDL



© Copyright IBM Corporation 2016

Figure 9-14. Generated WSDL

WM676/ZM6761.0

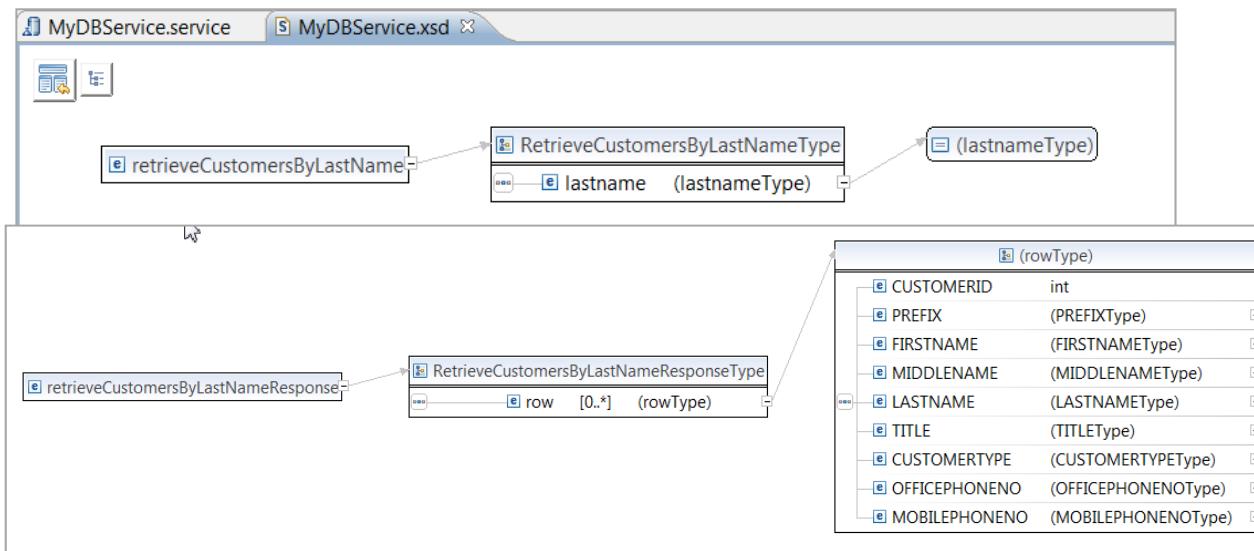
### Notes:

This figure shows an example WSDL that the Database Service editor generates.

The WSDL in the figure contains six operations:

- **retrieveCustomerByID**: Request/response service operation that is implemented as a database SELECT
- **retrieveCustomersByLastName**: Request/response service operation that is implemented as a database SELECT
- **retrieveCustomersInToronto**: Request/response service operation that is implemented as a database SELECT
- **createCustomer**: One-way service operation that is implemented as database CREATE
- **updateCustomer**: One-way service operation that is implemented as database UPDATE
- **deleteCustomer**: One-way service operation that is implemented as database DELETE

## Generated XSD



- Database Service editor generates an XML schema that describes the inputs and outputs of the database service

© Copyright IBM Corporation 2016

Figure 9-15. Generated XSD

WM676/ZM6761.0

### Notes:

The Database Service editor also generates an XML schema document that describes the inputs and outputs of the database operations.

You can click the service link on the Database Service editor **Summary** page to open the WSDL for the service. You can also click the link under the **Data Types** folder to open the XML schema file to review the input and output types for the operation.

The example in the figure shows the inputs and outputs for the **retrieveCustomerByLastName** operation and the **retrieveCustomersByLastNameResponse** operation.



## Database service in the Application Development view

Double-click the `.service` file or any of the operations to open the Database Service editor with the focus on the **Service Interface** tab

Review the logical interfaces and operations created in this service	
<b>Interface</b>	
Name	CUSTOMER_OPS_GROUP
Namespace	<a href="http://MyDBService">http://MyDBService</a>
<b>Operations</b>	
Operation	Message
<b>retrieveCustomerByID</b>	
<b>Input</b>	<a href="#">retrieveCustomerByID</a>
<b>Output</b>	<a href="#">retrieveCustomerByIDResponse</a>
<b>retrieveCustomersByLastName</b>	
<b>Input</b>	<a href="#">retrieveCustomersByLastName</a>
<b>Output</b>	<a href="#">retrieveCustomersByLastNameResponse</a>

© Copyright IBM Corporation 2016

Figure 9-16. Database service in the Application Development view

WM676/ZM6761.0

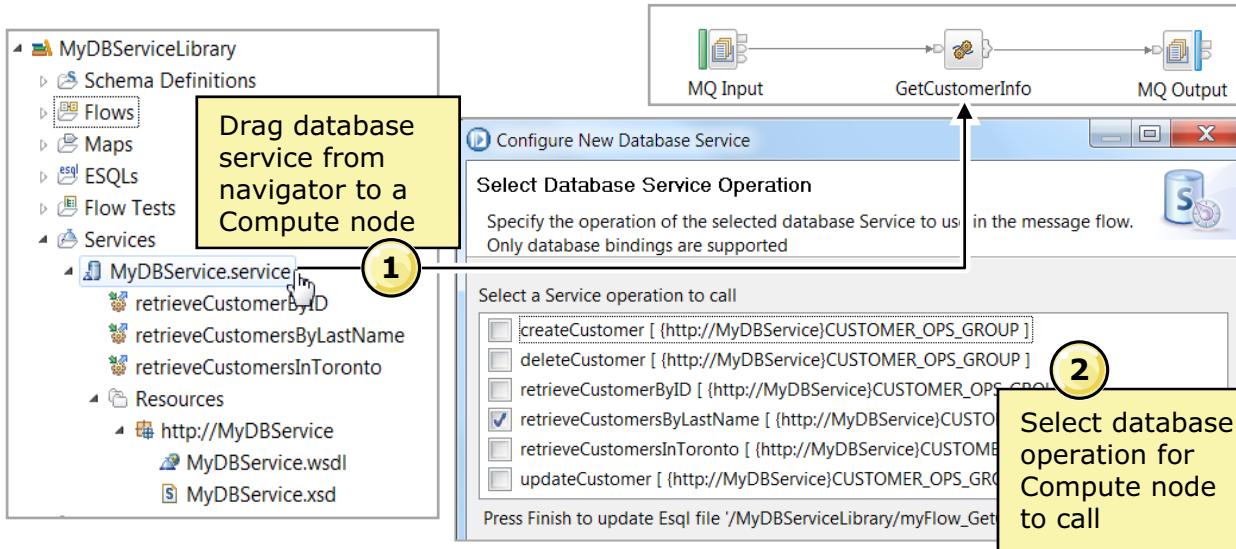
### Notes:

A service file (with a `.service` extension) captures metadata that is used during the discovery process. The Integration Toolkit uses this file to store state information from the Database Service editor, and iteratively discovers previously discovered artifacts.

When you save the database service, the artifacts are added to the **Application Development** view under the `<service_name>.service` file. The service artifacts include the operations, the WSDL, and the XSD.

You can open the Database Service editor and modify the service by double-clicking the `.service` file in the **Application Development** view.

## Calling a database service from a Compute node (1 of 2)



- Adds PATH statement in the ESQL file for the Compute module
  - Makes the service operation visible
  - Enables content assist on the operation
- Generates ESQL implementation code for the service operation in the `<DBService>.ESQL` file in the library under the ESQL package `<DBService>.<PortType>`

© Copyright IBM Corporation 2016

Figure 9-17. Calling a database service from a Compute node (1 of 2)

WM676/ZM6761.0

### Notes:

After you create the database service, you can use it to create the ESQL for a Compute node in a message flow.

1. Drag the database service (.service file) from the **Application Development** view onto the Compute node in the Message Flow editor. The Message Flow editor also supports a menu on the Compute node for selecting a database service.
2. Select the operation from the list of operations that are defined in the database service.

A compute module that is named `<DatabaseServiceName>.esql` is generated in the broker schema that is derived from the namespace of the service and port type of the database binding. The port type name is the same as the operations group name created on the **Service Operations** page of the Database Service editor. This file is in the library that contains the .service file, and contains an ESQL procedure definition for each database operation selected.

A PATH statement for the generated compute module is created in the ESQL module that is associated with the Compute node so that the database service can be called.

## Calling a database service from a Compute node (2 of 2)

The diagram shows a snippet of ESQL code within a code editor window titled "myFlow.msgflow" containing "myFlow\_GetCustomerInfo.esql". The code defines a compute module and function to call a database service operation "retrieveCustomersByLastName". Annotations explain the "PATH" statement, the output parameter "dbResultSetRef", and the call to the database service operation.

```

myFlow.msgflow  myFlow_GetCustomerInfo.esql
PATH MyDBService.CUSTOMER_OPS_GROUP; ← PATH statement

DECLARE ns NAMESPACE 'http://MyDBService';

CREATE COMPUTE MODULE myFlow_GetCustomerInfo
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();

    DECLARE LASTNAME CHAR;
    SET LASTNAME = InputRoot.XMLNSC.ns:retrieveCustomersByLastName.LASTNAME;

    -- Declare a location to hold the data returned from the database service call
    DECLARE dbResultSet ROW;
    DECLARE dbResultSetRef REFERENCE TO dbResultSet;
    -- Get the result set data returned from database service call from rowRef
    DECLARE rowRef REFERENCE TO dbResultSetRef.row;
    -- Call the database service operation retrieveCustomer in port type CUSTOMERGroup

    → CALL retrieveCustomersByLastName(LASTNAME, dbResultSetRef);
    SET OutputRoot.XMLNSC.ns:retrieveCustomersByLastNameResponse = dbResultSetRef;

    RETURN TRUE;
  END; ← Call database service operation as an ESQL function

```

**Output parameter dbResultSetRef points to the first occurrence of the row of the result set returned from the database service operation**

**Call database service operation as an ESQL function**

© Copyright IBM Corporation 2016

Figure 9-18. Calling a database service from a Compute node (2 of 2)

WM676/ZM6761.0

### Notes:

This figure shows an example of the ESQL routine that calls the database service.

The PATH statement at the top of the ESQL file identifies the database service.

The CALL statement calls the database service operation. In this example, the operation is `retrieveCustomersByLastName`. The CALL statement includes an input parameter (`LASTNAME`), and an output parameter (`dbResultSetRef`).

The output parameter, `dbResultSetRef`, points to the first occurrence of the row of the result set that is returned from the database service operation.



## Generated ESQL code for service operation

```

myFlow.msgflow myFlow_GetCustomerInfo.esql *MyDBService.esql
/*
 * Database Service operation retrieveCustomersByLastName:
 *
 * Data is returned through elements named 'row', created under INOUT parameter dbResultSetRef
 * of ESQL type ROW, containing:
 *   INTEGER CUSTOMERID
 *   VARCHAR PREFIX
 *   VARCHAR FIRSTNAME
 *   VARCHAR MIDDLENAME
 *   VARCHAR LASTNAME
 *   VARCHAR TITLE
 *   VARCHAR CUSTOMERTYPE
 *   VARCHAR OFFICEPHONENO
 *   VARCHAR MOBILEPHONENO
 *
 * Below is an example showing how to invoke the procedure for database service:
 *
 * -- Declare a location to hold the data returned from the database service call
 * DECLARE dbResultSet ROW;
 * DECLARE dbResultSetRef REFERENCE TO dbResultSet;
 *
 * -- Call the database service operation
 * CALL retrieveCustomersByLastName(lastname, dbResultSetRef);
 *
 * -- Get the result set data returned from database service call from rowRef
 * DECLARE rowRef REFERENCE TO dbResultSetRef.row;
 *
 */
CREATE PROCEDURE retrieveCustomersByLastName(IN lastname CHARACTER, INOUT dbResultSetRef REFERENCE)
BEGIN
  SET dbResultSetRef.row[] = PASSTHRU (
    'SELECT CUSTOMERID, PREFIX, FIRSTNAME, MIDDLENAME, LASTNAME, TITLE, CUSTOMERTYPE,
      OFFICEPHONENO, MOBILEPHONENO FROM SAMPLE.CUSTOMER
      WHERE SAMPLE.CUSTOMER.LASTNAME = ?
      ORDER BY SAMPLE.CUSTOMER.LASTNAME ASC, SAMPLE.CUSTOMER.FIRSTNAME ASC'
      TO Database.SAMPLEDB VALUES (lastname));
END;

```

© Copyright IBM Corporation 2016

Figure 9-19. Generated ESQL code for service operation

WM676/ZM6761.0

### Notes:

You can drag a database service operation from the **Application Development** view on to a Compute node to create an ESQL procedure that is based on the operation.

The procedure that is shown in this figure is the procedure that the ESQL in the previous figure calls. It is created automatically by dropping the service operation on to a Compute node.

In the example, an ESQL procedure is created for a service operation that is named `retrieveCustomersByLastName`. The service operation determines the input and output parameters for the operation.

The SELECT statement that is generated in the example in the figure is:

```

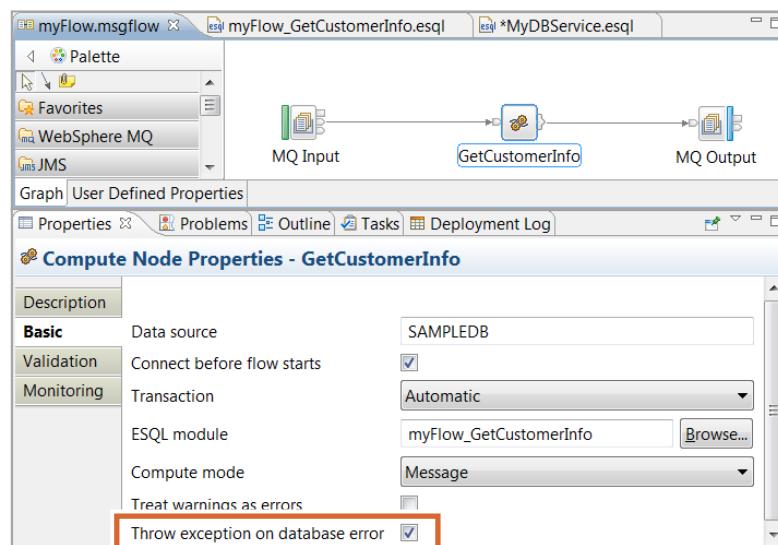
SET dbResultSetRef.row[ ] = PASSTHRU (
  'SELECT CUSTOMERID, PREFIX, FIRSTNAME, MIDDLENAME, LASTNAME, TITLE, CUSTOMERTYPE,
    OFFICEPHONENO, MOBILEPHONENO FROM SAMPLE.CUSTOMER
    WHERE SAMPLE.CUSTOMER.LASTNAME = ?
    ORDER BY SAMPLE.CUSTOMER.LASTNAME ASC, SAMPLE.CUSTOMER.FIRSTNAME ASC'
    TO Database.SAMPLEDB VALUES (lastname));

```

The ESQL generated in this example uses the ESQL PASSTHRU for the SELECT statement because it contains ORDER BY clause, which ESQL does not support.

## Database service operation error handling

- Error handling is identical to standard practice for working with databases that use ESQL on Compute node
- **Throw exception on database error** property
  - Errors from calling database service are raised as ESQL exceptions
  - Enabled by default
  - If not selected, then after calling the database service function, check for errors by using SQLCODE, SQLSTATE, SQLERRORTEXT, and SQLNATIVEERROR



© Copyright IBM Corporation 2016

Figure 9-20. Database service operation error handling

WM676/ZM6761.0

### Notes:

The error handling options for a database service operation are the same as any ESQL that references a database on a Compute node.

One option is to add ESQL code to handle database errors that might occur when you call the generated procedure.

Another option is to select the option to **Throw exception on database error** on the Compute node properties, which is enabled by default.

Database error handling is described in detail in the prerequisite course WM666, *IBM Integration Bus V10 Application Development I*.

## Unit summary

Having completed this unit, you should be able to:

- Create a database service
- Use the database service to populate node properties

© Copyright IBM Corporation 2016

Figure 9-21. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or false: You can reference only one table in a database service.
2. True or false: Database services can be published to the Integration Registry.

© Copyright IBM Corporation 2016

---

Figure 9-22. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.



## Checkpoint answers

1. True or false: You can reference only one table in a database service.

Answer: **True**.

2. True or false: Database services can be published to the Integration Registry.

Answer: **False**. Database services cannot be published to the Integration Registry.

© Copyright IBM Corporation 2016

Figure 9-23. Checkpoint answers

WM676/ZM6761.0

### Notes:



# Unit 10. Connecting IBM MQ by using a discovered service

## What this unit is about

An IBM MQ service defines the interaction between IBM Integration Bus and IBM MQ applications. In this unit, you learn how to define an IBM MQ service. You also learn how to publish the IBM MQ service in the IBM Integration Bus Integration Registry and implement the service in a message flow.

## What you should be able to do

After completing this unit, you should be able to:

- Create an IBM MQ request and response service
- Publish the IBM MQ services in an Integration Registry
- Create a message flow that implements an IBM MQ service

## How you will check your progress

Checkpoints

Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus V10



## Unit objectives

After completing this unit, you should be able to:

- Create an IBM MQ request and response service
- Publish the IBM MQ services in an Integration Registry
- Create a message flow that implements an IBM MQ service

© Copyright IBM Corporation 2016

---

Figure 10-1. Unit objectives

WM676/ZM6761.0

### Notes:



## IBM MQ service

- Defines the interaction between Integration Bus and IBM MQ applications
- Use to configure the properties of an IBM MQ node in a message flow and create service subflows
- Contains:
  - WSDL file that contains connection information about the queue manager and resulting queue names
  - XSD that describes service input and output
  - XML file that contains metadata that is used during the service discovery process
- Option to automatically create an MQEndpoint service and store it in an Integration Registry when you save the service definition

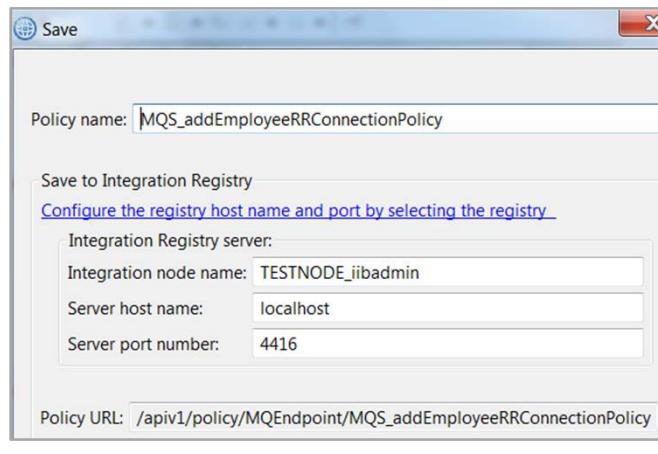


Figure 10-2. IBM MQ service

WM676/ZM6761.0

### Notes:

An IBM MQ service defines the interaction between Integration Bus and IBM MQ applications. You can use an IBM MQ service to discover artifacts from queue managers. You can then use these artifacts in a message flow.

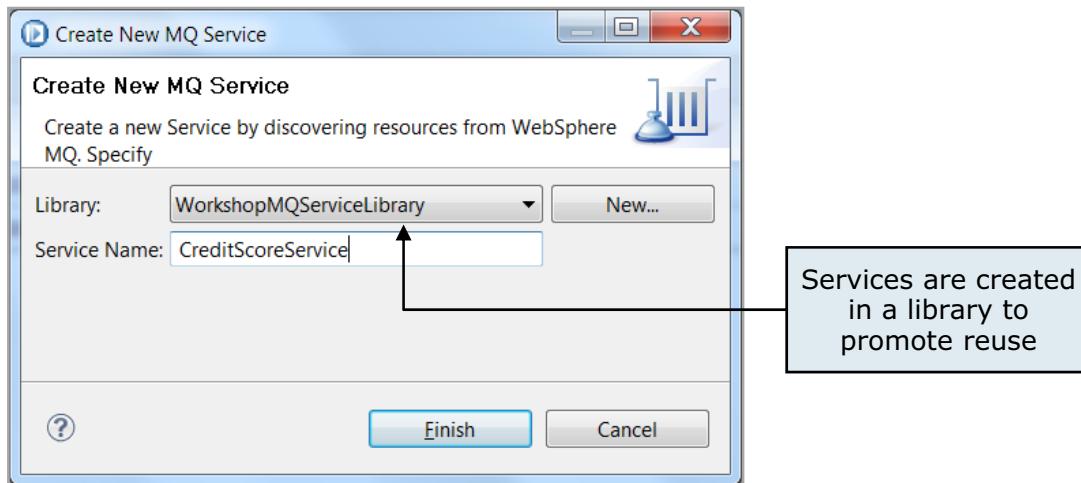
The IBM MQ Service definition specification describes the resources that you discover. An IBM MQ service contains:

- A WSDL file that contains connection information about the queue manager and resulting queues.
- An XSD file for the input message, and an XSD file for the output message, if any are specified in the IBM MQ Service editor. The XSD files contain XML structures that describe service input and output.
- An XML file that contains metadata that is used during the discovery process. The Integration Toolkit uses this file to store state information from the IBM MQ Service editor, and iteratively discover previously discovered artifacts.



## Starting the IBM MQ Service editor

- Click **File > New > MQ Service** or
- Click **Start by discovering a service** from the **Quick Start** menu



© Copyright IBM Corporation 2016

Figure 10-3. Starting the IBM MQ Service editor

WM676/ZM6761.0

### Notes:

You can create an IBM MQ service definition in the Integration Toolkit by clicking **File > New > MQ Service** or by clicking **Start by discovering a service** from the **Quick Start** menu.

When you create a service, you must specify a library name and a service name.

A service definition file is created for each IBM MQ service that you discover. An IBM MQ service definition file has an extension of .service.



## IBM MQ Service editor: Description

**MQService.service**

**Checklist**

- Description**
- Connect
- Select Resources
- MQ Configuration
  - Request Message Headers
  - Response Message Headers
- Service Interface
- Summary

You can use an MQ Service to define the interaction between Application and MQ service.

This MQ Service editor helps you discover queues that exist in the system and to define a WSDL file to use with the IBM MQ Service definition.

**Application**

Legend: Logical (grey box) Physical (yellow box)

Description page describes the steps for creating the service and the artifacts that the service defines

After you create an MQ Service, you can use it to create and edit the Message Flow editor, or onto an MQInput node or MQOut node.

Click [here](#) for additional information on MQ Services.

© Copyright IBM Corporation 2016

Figure 10-4. IBM MQ Service editor: Description

WM676/ZM6761.0

### Notes:

The IBM MQ Service editor contains seven pages that the **Checklist** identifies. You can click the option in the **Checklist** to go directly to a page.

The IBM MQ Service editor opens on the **Description** page. The **Description** page describes the steps for defining an IBM MQ service and the artifacts that the IBM MQ Service editor generates.




## IBM MQ Service editor: Connect

Specify connection parameters for the remote system to discover resources

**i** Connection was tested successfully

Connection:	Local queue manager
Queue Manager Name:	IIBQM
CCDT file URL:	
Queue Manager Host Name:	localhost
Listener Port Number:	1414
Channel Name:	SYSTEM.BKR.CONFIG
Security Identity:	
Use SSL:	<input type="checkbox"/>
SSL Peer Name:	
SSL Cipher Specification:	
<a href="#">Test Connection</a>	

- Discover queues on local or remote queue manager
- Define connection options
  - **Local binding connection:** Connect to local queue manager
  - **Remote client connection:** Use client channel definition table (CCDT) file or specify the target queue manager by using host name (IP), port, and channel
- Option to test the connection

© Copyright IBM Corporation 2016

Figure 10-5. IBM MQ Service editor: Connect

WM676/ZM6761.0

### Notes:

You define the local or remote queue manager on the **Connect** page. The IBM MQ Service editor connects to the specified queue manager, either locally or remotely, to discover all the existing queues on the queue manager.

If you specify **Remote client connection** in the **Connection** field, the options for identifying the remote queue manager are available. You can identify the remote queue manager by selecting a client channel definition table or by specifying an IP address or host name.

Click the **Test Connection** link after you identify the queue manager to ensure that the Integration Toolkit can connect to the queue manager.

The screenshot shows the 'IBM MQ Service editor: Select Resources' page. At the top, there's a navigation bar with 'WebSphere Education' and the IBM logo. Below it, the title 'IBM MQ Service editor: Select Resources' is displayed. The main area has a breadcrumb navigation: 'CreditScoreService.service > 1. Description > 2. Connect > 3. Select Resources > 4. MQ Configuration > 5. Service Interface > 6. Summary'. A note says 'Query and select the resources from the target system'. It includes sections for 'Message Exchange Pattern' (radio buttons for 'Select resources for Request-Response operation' and 'Select resources for One-Way operation'), 'Filter Text' (text input field with placeholder '\*'), 'Include Resources' (checkboxes for 'System Queues' and 'Dynamic Queues'), and a 'Queues' list box containing various queue names like 'CreditScore\_In', 'CreditScore\_Out', 'In', etc. To the right of the queues list, there are buttons for 'Select Request' (with input field 'CreditScore\_In') and 'Select Response' (with input field 'CreditScore\_Out'). A yellow callout box contains the text: 'On the **Select Resources** page, select the service operation type and the queues for request and response'. Navigation links at the bottom include '< Previous' and 'Next >'. A copyright notice at the bottom right reads '© Copyright IBM Corporation 2016'.

Figure 10-6. IBM MQ Service editor: Select Resources

WM676/ZM6761.0

## Notes:

On the **Select Resources** page, you can define a request/response operation or a one-way operation. If you click **Select resources for Request-Response operation**, you select a request queue and response queue. You create a request/response operation in the exercise for this unit.

To reduce the number of queues that are displayed in the **Queues** list, specify the filter text. Optionally, you can also choose to display the system queues, dynamic queues, or both.

You can click **Refresh queues** to view any queues that you added after you set the parameters on the **Connect** page. All existing queues are shown.




## IBM MQ Service editor: MQ Configuration

**Request Message Headers**

The configured MQ headers can be used to configure the request message when the message flow puts the message on the request queue.

CCSID:	<input type="text"/>
Format:	<input type="text"/>
Message type:	<input type="text"/> Request
Persistence:	<input type="text"/>
Message ID:	<input type="text"/>
Correlation ID:	<input type="text"/>
Expiry:	<input type="text"/>
Priority:	<input type="text"/>

**Response Message Headers**

The configured MQ headers can be used to configure the response message when the message flow implementing the Request-Reply pattern puts the message on the response queue.

CCSID:	<input type="text"/>
Format:	<input type="text"/>
Message type:	<input type="text"/> Reply
Persistence:	<input type="text"/>
Message ID:	<input type="text"/>
Correlation ID:	<input type="text"/>
Expiry:	<input type="text"/>
Priority:	<input type="text"/>

Click **MQ Configuration > Request Message Headers** in the **Checklist** to configure MQMD header for request messages

Click **MQ Configuration > Response Message Headers** in the **Checklist** to configure MQMD header for response messages

© Copyright IBM Corporation 2016

Figure 10-7. IBM MQ Service editor: MQ Configuration

WM676/ZM6761.0

### Notes:

On the **MQ Configuration** pages, you can optionally configure the IBM MQ message descriptor (MQMD) for the request message and the response message.

If you are creating a request/response service, complete the fields on both the **Request Message Headers** and **Response Message Headers** pages.

The screenshot shows the IBM MQ Service editor interface. At the top, there's a header with the WebSphere Education logo and the IBM logo. Below the header, the title "IBM MQ Service editor: Service Interface" is displayed. The main area contains a form for configuring a service interface. It includes sections for "Interface" (Name: CreditScoreService\_PortType, Namespace: http://tempuri.org/CreditScoreService, Message Configuration: Use XML schema types selected) and "Operations" (Operation: getCreditScore, Input message: creditScoreReq, Output message: creditScoreResp). A yellow callout box with a black border and arrow points to the "creditScoreReq" link, containing the text: "Click **Open file** to open the XSD schema file and review the contents of the request/response types for the operation".

- Specify service operation name and its input and output
- Input and output style can be either:
  - Use XML schema elements:** Global element (message name) is defined in the schema
  - Use XML schema types:** Global elements are named after the operation is created  
`<operationName>` for request message  
`<operationName>Response` for response message

© Copyright IBM Corporation 2016

Figure 10-8. IBM MQ Service editor: Service Interface

WM676/ZM6761.0

### Notes:

On the **Service Interface** page, you configure the interface and advanced binding parameters. If the service is not saved, clicking **Open file** opens a dialog that prompts you to save the service file. Saving the service file generates the WSDL and XSD files for the service.

**WebSphere Education**

**IBM MQ Service editor: Summary**

CreditScoreService.service

1. Description > 2. Connect > 3. Select Resources > 4. MQ Configuration > 5. Service Interface > 6. Summary

Review the Service structure and the list of files that are generated when you save a Service

Interface

- CreditScoreService
  - CreditScoreService\_PortType
    - getCreditScore
      - creditScoreReq
      - creditScoreResp
- Data Types
  - CreditScore
  - CreditScore

Click to open WSDL file for service

Click to open the XSD file to review the types for the operation

You can drag the generated Service onto the corresponding node on the Message Flow editor. This action configures the node so that it calls the discovered system. To make additional edits to the elements of the Service, use the service editor

© Copyright IBM Corporation 2016

Figure 10-9. IBM MQ Service editor: Summary

WM676/ZM6761.0

### Notes:

The **Summary** page includes the summary information about the operations and input and output message types.

Review the service structure and the list of files that are created when the service is saved. You can preview the WSDL by clicking the service link. You can preview the XSD file created for the service by clicking the link under **Data Types**.

Be sure to save the services by clicking **File > Save**.



## IBM MQ Service WSDL and XSD file

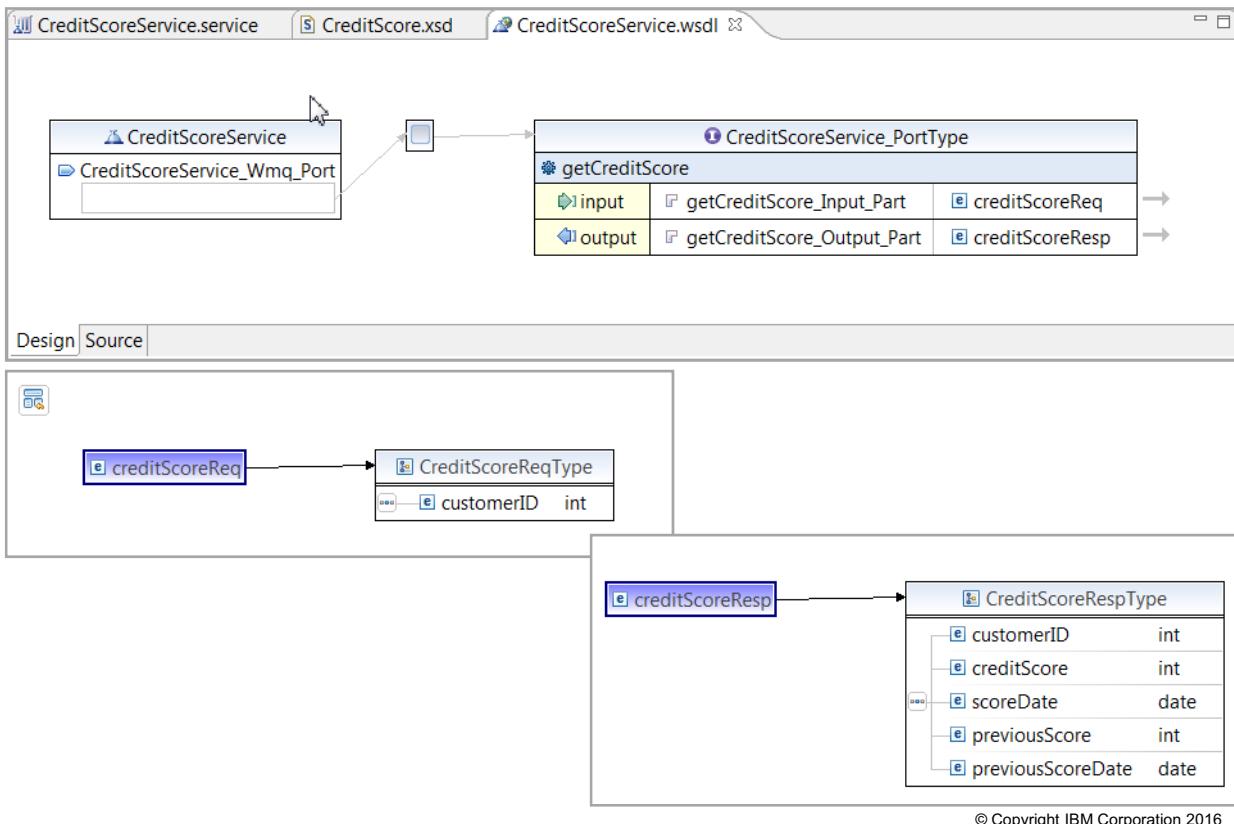


Figure 10-10. IBM MQ Service WSDL and XSD file

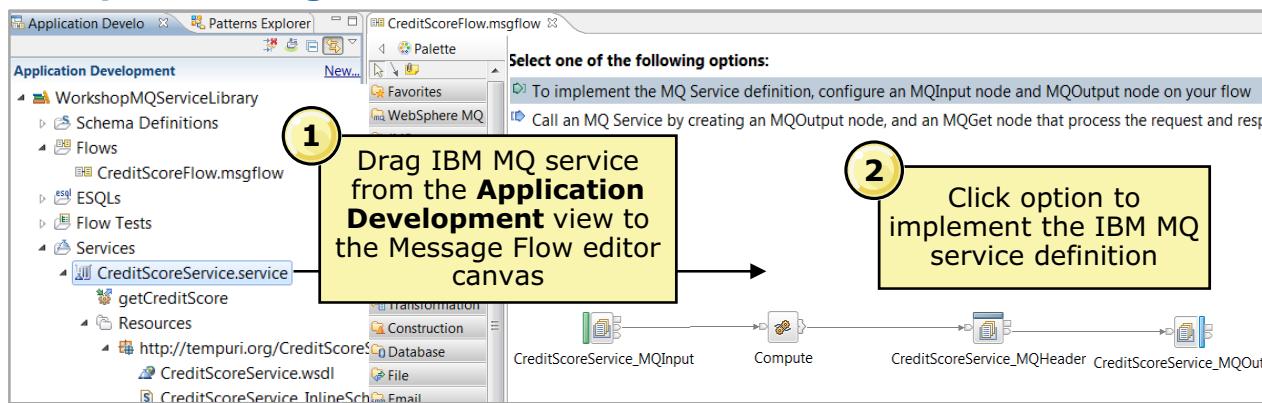
WM676/ZM6761.0

### Notes:

This figure shows the design view of the WSDL and the XSD that the IBM MQ Service editor generates.

# WebSphere Education

## Implementing an IBM MQ service



- Nodes added to flow canvas:
  - MQ Input node to receive a request
  - MQ Header node to prepare the reply header
  - MQ Output node to send a reply
- Node properties, such as queue name and header properties, get property values from the service definition
- Wire the nodes between the MQ Input and MQ Header node to implement the service

**Note:** Using the IBM MQ service to create the message flow requires IBM Integration Bus Toolkit V10.0.0.2 or later

© Copyright IBM Corporation 2016

Figure 10-11. Implementing an IBM MQ service

WM676/ZM6761.0

### Notes:

You can use an IBM MQ service to configure an IBM MQ message processing node in several ways.

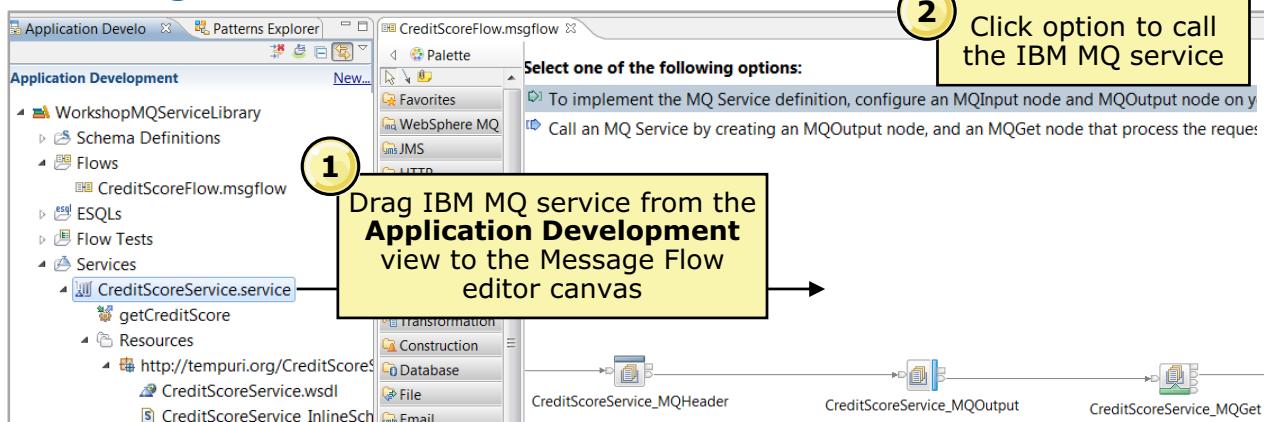
- Add an IBM MQ node in the Message Flow editor, and drag an IBM MQ service onto the node. The node is configured according to the IBM MQ service.
- Drag an IBM MQ service onto the Message Flow editor to implement an IBM MQ service (shown in the figure).
- Create an IBM MQ node in the Message Flow editor. Use the **Properties** view to configure a queue that is described in the IBM MQ service.

Implementing an IBM MQ service creates three nodes in the message flow:

- An MQ Input node that receives the request
- An MQ Header node to prepare the reply header
- An MQ Output node to send the reply



## Calling an IBM MQ service



- Nodes added to flow canvas:
  - MQ Header node to prepare the request header
  - MQ Output node to send a request
  - MQ Get node to receive a reply
- Node properties such as queue name, header properties, and transaction mode get property values from the service definition and IBM MQ request/reply pattern

**Note:** Using the IBM MQ service to create the message flow requires IBM Integration Bus Toolkit V10.0.0.2 or later

© Copyright IBM Corporation 2016

Figure 10-12. Calling an IBM MQ service

WM676/ZM6761.0

### Notes:

To create a message flow that calls an IBM MQ service:

1. Drag the IBM MQ service from the **Application Development** view to the Message Flow editor.
2. Click the option to call the IBM MQ service.

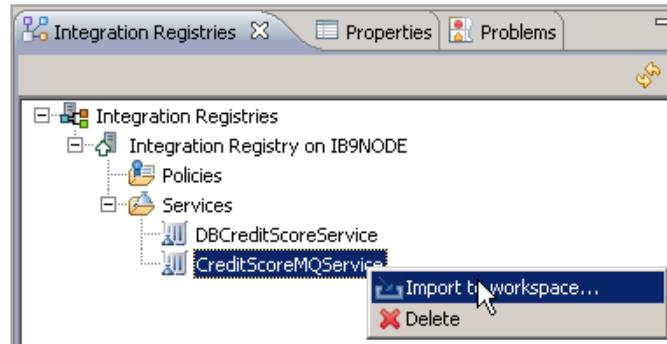
When you select the option to call a service, three nodes are created:

- An MQ Header node that prepares the request header
- An MQ Output node that sends the request
- An MQ Get node that receives the reply

 WebSphere Education 

## Integration Registry

- Contains service descriptions and workload management policies
  - Facilitates collaboration and reuse
  - Hosted by the IBM Integration Bus integration node
- Populated by:
  - Publishing discovered IBM MQ services in the Integration Toolkit
  - Publishing IBM MQEndpoint policies in the Integration Toolkit or the Integration web interface
  - Publishing workload management policies in the Integration web interface
- Can be accessed in the Integration Toolkit and the Integration web interface



© Copyright IBM Corporation 2016

Figure 10-13. Integration Registry

WM676/ZM6761.0

### Notes:

IBM Integration Bus provides an Integration Registry, which can store IBM MQ and MQTT service definitions and workload management policies.

The Integration Registry is hosted in a runtime integration node. Every runtime integration node can host one Integration Registry. All integration nodes that are created in Integration Bus have an Integration Registry available (enabled) by default.

The Integration Registry consists of the following parts:

- A model-based registry that provides a service provider catalog
- A file repository for storing text or binary documents

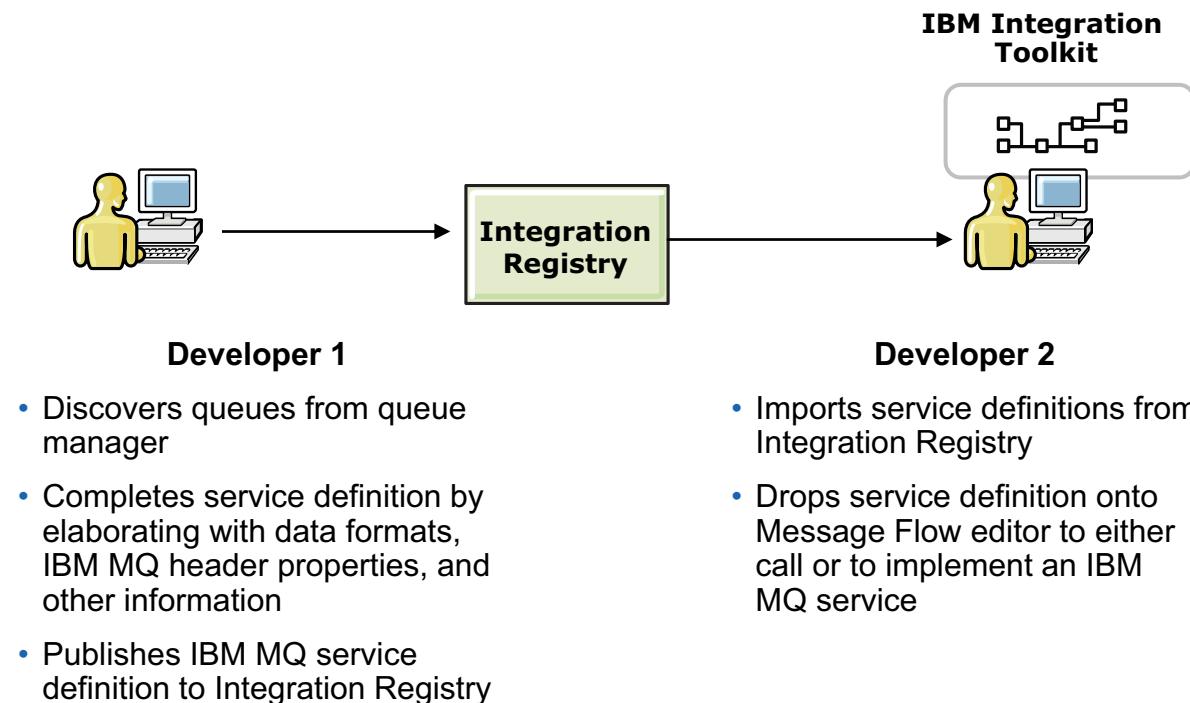
A URL uniquely identifies all artifacts that are created in the Integration Registry.

The Integration Registry can be accessed in the Integration Toolkit and the Integration web interface.

The Integration Registry was introduced in the prerequisite course, WM666: *IBM Integration Bus V10 Application Development I*.



## Discover and catalog services



© Copyright IBM Corporation 2016

Figure 10-14. Discover and catalog services

WM676/ZM6761.0

### Notes:

As illustrated in the figure, the Integration Registry allows the services that one developer creates to be used by another developer.

In the example, Developer 1 develops and publishes the IBM MQ service definition to the Integration Registry. Then, Developer 2 imports the service definition from the Integration Registry and uses the service to create a message flow.



## Publishing a service to the Integration Registry

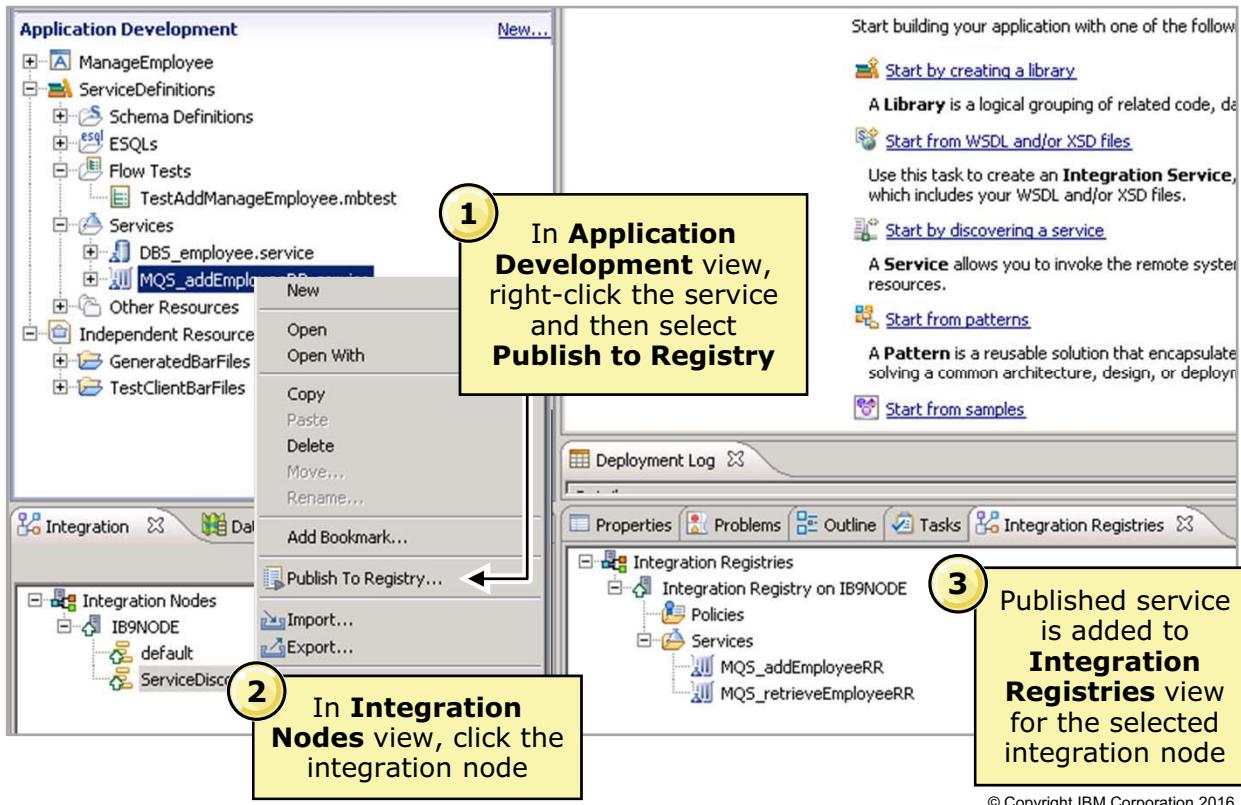


Figure 10-15. Publishing a service to the Integration Registry

WM676/ZM6761.0

### Notes:

Complete the following steps to publish your locally stored IBM MQ services to the Integration Registry:

1. In the **Application Development** view, right-click the IBM MQ service, and then click **Publish to Registry**.
2. In the **Publish to Registry** window, click the node that contains the Integration Registry that you want to publish. Click **Finish**.
3. In the **Integration Registries** view, click **Refresh** to update the list of IBM MQ services. The refreshed list includes the IBM MQ service that you added.



## Importing a service from the Integration Registry

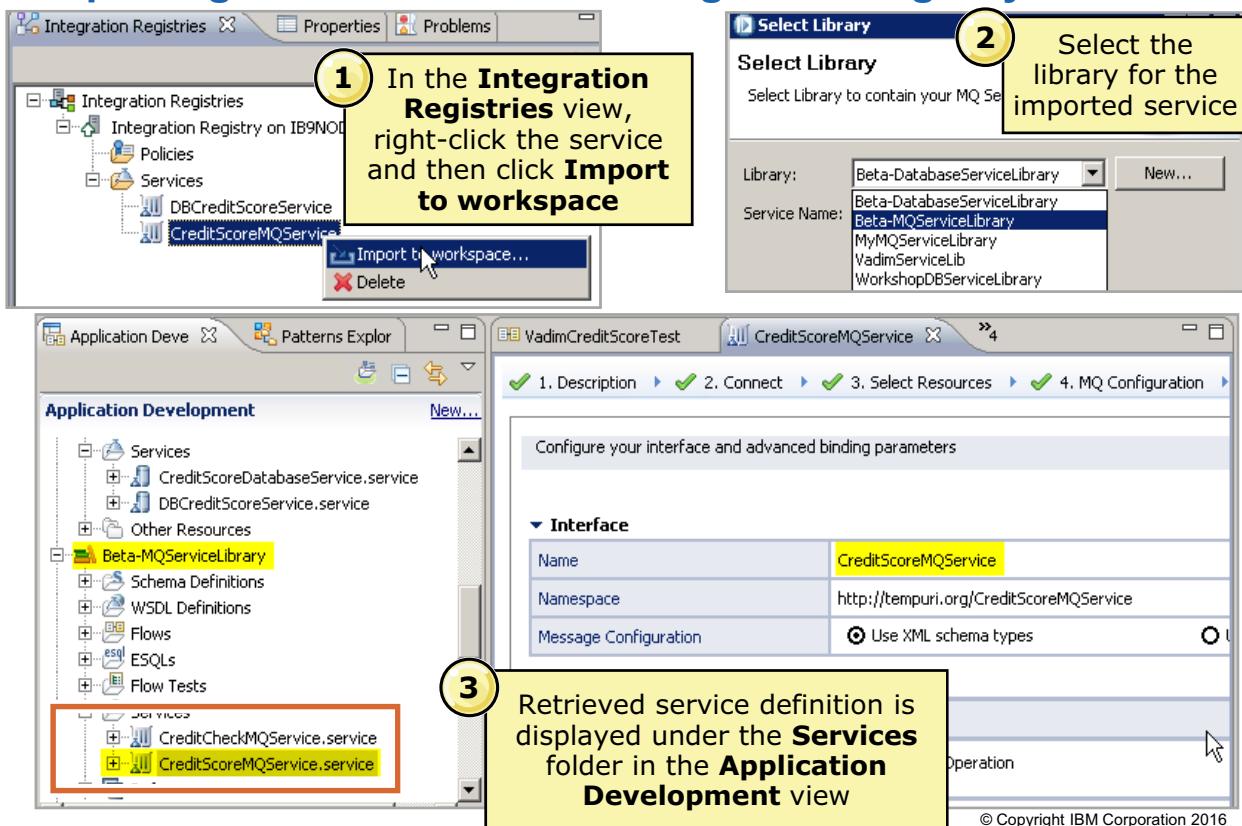


Figure 10-16. Importing a service from the Integration Registry

WM676/ZM6761.0

### Notes:

You can import an IBM MQ service that is stored in the Integration Registry into a new or existing project. You can then use the IBM MQ service in a message flow.

Complete the following steps to import an IBM MQ Service into a new or existing library:

1. In the **Integration Registries** view, right-click the IBM MQ service that you want to import, and then click **Import To Workspace**. The Select Library wizard opens.
2. Select the library that you want to contain the IBM MQ service or click **New** to create a library. Click **Finish**.
3. The service definition is displayed under the **Services** folder in the specified library.



## Unit summary

Having completed this unit, you should be able to:

- Create an IBM MQ request and response service
- Publish the IBM MQ services in an Integration Registry
- Create a message flow that implements an IBM MQ service

© Copyright IBM Corporation 2016

---

Figure 10-17. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or false: IBM MQ and database services can be published to the Integration Registry.
2. True or false: You can add operations to an IBM MQ service after it is created.

© Copyright IBM Corporation 2016

Figure 10-18. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.



## Checkpoint answers

1. True or false: IBM MQ and database services can be published to the Integration Registry.

Answer: **False**. Database services cannot be published to the Integration Registry.

2. True or false: You can add operations to an IBM MQ service after it is created.

Answer: **False**. You can add operations to a database service after it is created, but not an IBM MQ service.

© Copyright IBM Corporation 2016

Figure 10-19. Checkpoint answers

WM676/ZM6761.0

### Notes:

## Exercise 5



Creating IBM MQ and database services

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 10-20. Exercise 5

WM676/ZM6761.0

### Notes:

## Exercise objectives

After completing this exercise, you should be able to:

- Create an IBM MQ request and response service definition
- Publish the IBM MQ services in an Integration Registry
- Import previously stored IBM MQ services into a developers workspace
- Create a database service
- Extend an existing database service by adding more operations

© Copyright IBM Corporation 2016

---

Figure 10-21. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.

# Unit 11. Creating a decision service

## What this unit is about

A Decision Service message processing node runs business rules to provide routing, validation, and transformation operations. You can write rules in the IBM Integration Toolkit, or import rules from IBM Operational Decision Manager. In this unit, you learn how to create a decision service that implements business rules and then use the decision service in a message flow application to provide routing, validation, and transformation.

## What you should be able to do

After completing this unit, you should be able to:

- Create a decision service that implements business rules to provide routing, validation, and transformation

## How you will check your progress

Checkpoints

Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus V10

## Unit objectives

After completing this unit, you should be able to:

- Create a decision service that implements business rules to provide routing, validation, and transformation

© Copyright IBM Corporation 2016

Figure 11-1. Unit objectives

WM676/ZM6761.0

### **Notes:**



## IBM Operational Decision Manager

- Manage business policies
- Enable social collaboration to manage business change
- Apply IBM Operational Decision Manager
  - To flexibly and reliably manage repeatable, automated decisions
  - When decisions change frequently
  - To increase straight-through processing
  - When decision services must be shared across systems
  - To manage and govern large numbers of rules
  - When real-time events require immediate actions

© Copyright IBM Corporation 2016

Figure 11-2. IBM Operational Decision Manager

WM676/ZM6761.0

### Notes:

IBM Operational Decision Manager is a full-featured platform for capturing, automating, and governing frequent, repeatable business decisions. It consists of two components, IBM Decision Center and IBM Decision Server. These components form the platform for managing and running business rules and business events to help you make decisions faster, improve responsiveness, minimize risks, and seize opportunities.

Operational Decision Manager improves the quality of transaction and process-related decisions. It helps determine the appropriate course of action for customers, partners, and internal interactions. Operational Decision Manager improves business insight and outcomes and helps you detect opportunities and risks. You can also implement, test, and deploy decision changes and understand how decisions are made and apply them consistently across processes and applications.

## Decision services in Integration Bus

- Use business metrics and message content to provide business level routing in message flows
  - Choose the destination that imposes the least cost to the business while maintaining service level agreements
  - Analyze the message content and route to the destination best suited to fulfill the request
- Perform complex validation of content of large documents based on business defined rules
- Augment message content with information stored in decision tables
- Decision Service node runs business rules
  - Write rules in the Integration Toolkit
  - Import rules from Operational Decision Manager
  - Retrieve rules from an external Operational Decision Manager database repository

© Copyright IBM Corporation 2016

Figure 11-3. Decision services in Integration Bus

WM676/ZM6761.0

### Notes:

With the Decision Service node, Integration Bus can call business rules that run on a component of IBM Decision Server that is provided with IBM Integration Bus.

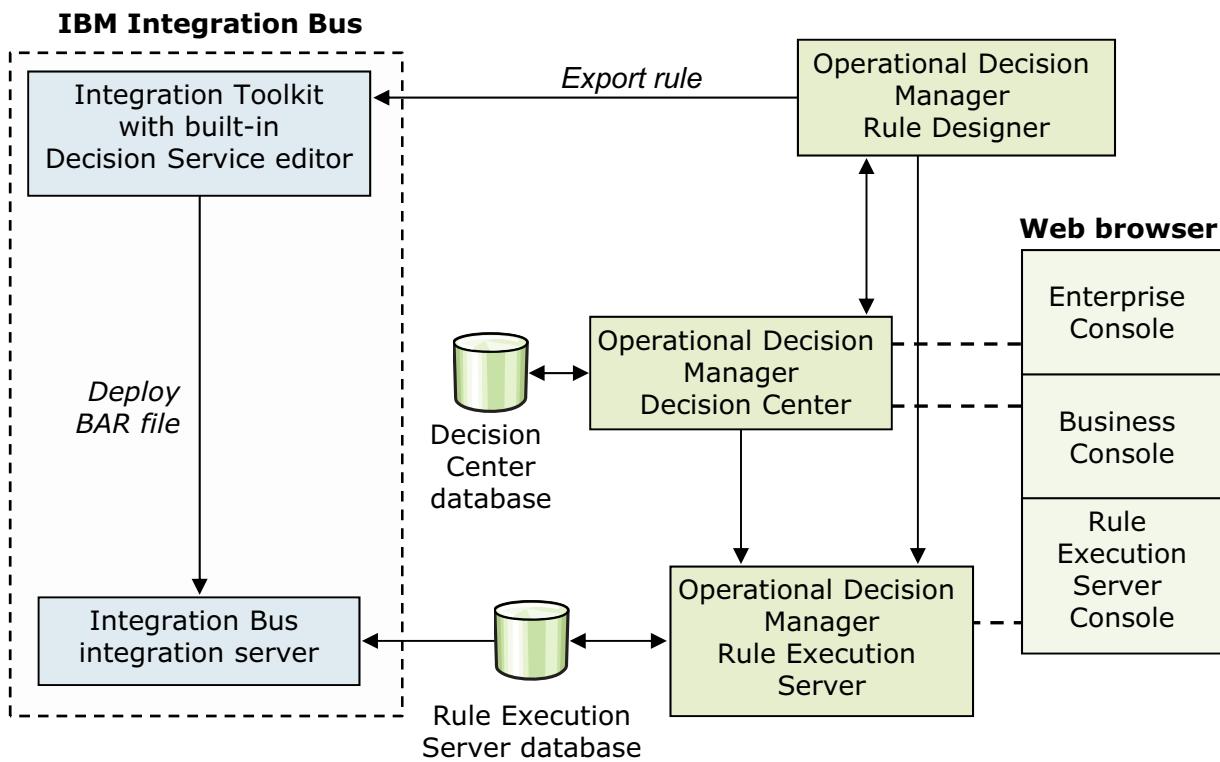


#### Important

The IBM Integration Bus license entitles you to use the IBM Decision Server only through the Decision Service node and only for development and functional test. To use the IBM Decision Server beyond development and functional test, you must purchase a separate license for either IBM Decision Server or IBM Decision Server Rules Edition for Integration Bus.

You can also use rules that are written in Operational Decision Manager to process messages in Integration Bus. You can import a rule application archive from Operational Decision Manager and deploy those business rules at run time. You can also retrieve the same business rules at run time from an external Operational Decision Manager repository.

## Integration with Operation Decision Manager



© Copyright IBM Corporation 2016

Figure 11-4. Integration with Operation Decision Manager

WM676/ZM6761.0

### Notes:

The figure shows the primary Operational Decision Manager components that are needed to define rules within the Operational Decision Manager system, and to deploy and load those rules into Integration Bus.

The Decision Center provides an integrated repository and management components. It allows subject matter experts to maintain and govern their business decisions.

The Decision Server provides the runtime components to automate decision logic. It enables the detection of business situations and precise response based on the context of the interaction.

The Integration Bus runtime loads rules directly from the Operational Decision Manager database. Integration Bus can automatically detect and load updates to those rules at run time.

## Structure of a business rule

- Definitions** set parameters that identify business terms by using easily understandable names.

Example:

```
definitions
  set minimum_cart_value to $1500
```

- Condition statements** define the conditions under which actions are completed.

- If the condition is true, the action is completed
- A rule can contain one or more condition statements

Example:

```
if
  the customer's category is Gold
  and the value of the customer's shopping cart is more
  than minimum_cart_value
```

- Action statements** define the actions to take when the conditions that **if** statements represent are met.

- If more than one action, the actions are taken in the order in which they are written
- Rule can also contain **else** statements in the actions section

Example:

```
then
  change the customer's category to Platinum
```

© Copyright IBM Corporation 2016

Figure 11-5. Structure of a business rule

WM676/ZM6761.0

### Notes:

A business rule is a required operation that applies to a specific set of business conditions. For example, you can create a business rule that offers a discount to customers who spend more than a certain amount. The business rule can be modified in the future if the business climate changes and the amount of discount must change.

You can use Integration Bus to write business rules by using natural language instead of coding in ESQL or Java. The use of natural language in the authoring of the rules means that business users, such as a business analyst, can understand them.

A business rule typically consists of the following information, in the specified order:

- Definitions: At the beginning of the rule, you can set parameters that identify business terms by using easily understandable names.
- Conditions: The conditions section of the rule contains the “if” statements. These statements define the conditions under which actions are completed. If the condition is true, the action is completed. A rule can contain one or more condition statements.
- Actions: The actions section of the rule contains the “then” statements. These statements define the actions that are taken when the conditions that the “if” statements represent are met. If the

actions section contains more than one action, the actions are taken in the order in which they are written. You can also include “else” statements in the actions section. These statements define what actions to take when the conditions are not met.



## Steps for implementing a decision service

1. Define a decision service by using the **Create a Decision Service** wizard.
  - Required schema files must be available in the application or in an associated library
  - Select XMLNSC or DFDL schemas
2. Use Content Assist editor to author business rules.
3. Embed the decision service in the message flow.
4. Configure the Decision Service node.
  - Specify the location of the data in the logical message tree on the **Basic** properties

© Copyright IBM Corporation 2016

Figure 11-6. Steps for implementing a decision service

WM676/ZM6761.0

### Notes:

This figure lists the steps for implementing a decision service. These steps assume that you are using the Rules editor in the Integration Toolkit. Each of these steps is described in more detail and in an example later in this unit.



## Integration Toolkit Decision Service editor

Use the table below to customize the parameters available to the decision service.

Name	Type	Direction	Verbalization
StockTrade	StockTrade {http://www.ibm.com/demo}	INOUT	the stock trade

**Decision Service Parameters**

Add parameters to a decision service

Rename parameters so that they are more descriptive

Rule sequence Parameters Additional Info

- Use the **Rule sequence** tab to add rules and specify the order in which they run
- Use the **Parameters** tab to modify the **Name** and **Verbalization** values and to add, edit, and delete parameters
- Use the **Additional Info** tab to specify the version of the decision service and to provide a description

© Copyright IBM Corporation 2016

Figure 11-7. Integration Toolkit Decision Service editor

WM676/ZM6761.0

### Notes:

When you create a decision service in the Integration Toolkit, it contains a list of the schema parameters that you select with the wizard. You can configure those parameters, provide natural language aliases for them, and use the content assist tool to write business rules.

In the Decision Service editor, use the **Rule sequence** tab to add rules and specify the order in which they are run. Use the **Parameters** tab to add, edit, and delete parameters. Use the **Additional Info** tab to specify the version of the decision service and to provide a description.



#### Note

If your decision service is created from a rule application archive that is imported from IBM Operational Decision Manager, you cannot add, edit, or delete parameters. Nor can you see or edit the rules that are contained in the rule set. When you open an imported decision service in the Decision Service editor, you see only the **Parameters** and **Additional Info** tabs, and you cannot edit the information that is on these tabs.

## Decision Service node

- Implements business rules to provide operations like routing, validation, and transformation
- Allows Integration Bus to call business rules that run on a component of IBM Decision Server that is provided with Integration Bus for development and functional testing
- Created by dragging a decision service from the Integration Toolkit **Application Development** view onto the Message Flow editor canvas
- Before you deploy message flows that contain a Decision Service node, you must enable the mode extension by using the `mqsimode` command: **mqsimode -x DecisionServices**

To use the IBM Decision Server component beyond development and functional test, you must purchase a separate license entitlement for either IBM Decision Server or IBM Decision Server Rules Edition for Integration Bus

© Copyright IBM Corporation 2016

Figure 11-8. Decision Service node

WM676/ZM6761.0

### Notes:

A Decision Service message processing node implements business rules to provide operations like routing, validation, and transformation. These business rules are organized in a decision service file (.rules). When you deploy the BAR file that contains your decision service, the decision service is compiled into a rule set (.ruleset). You can store a decision service in an application, library, or integration project.

If you create the decision service, you can drag the decision service onto a Decision Service node in the Message Flow editor to configure the node. You can also associate an existing decision service with a node by using the **Decision Service** property of the node.

To use the IBM Decision Server component beyond development and functional test, you must purchase a separate license entitlement for either IBM Decision Server or IBM Decision Server Rules Edition for Integration Bus. Before you deploy a message flow that contains a decision service, confirm that you comply with the terms of the license. Then, enable the decision services mode extension by using the `mqsimode` command. The following example can be used on distributed systems:

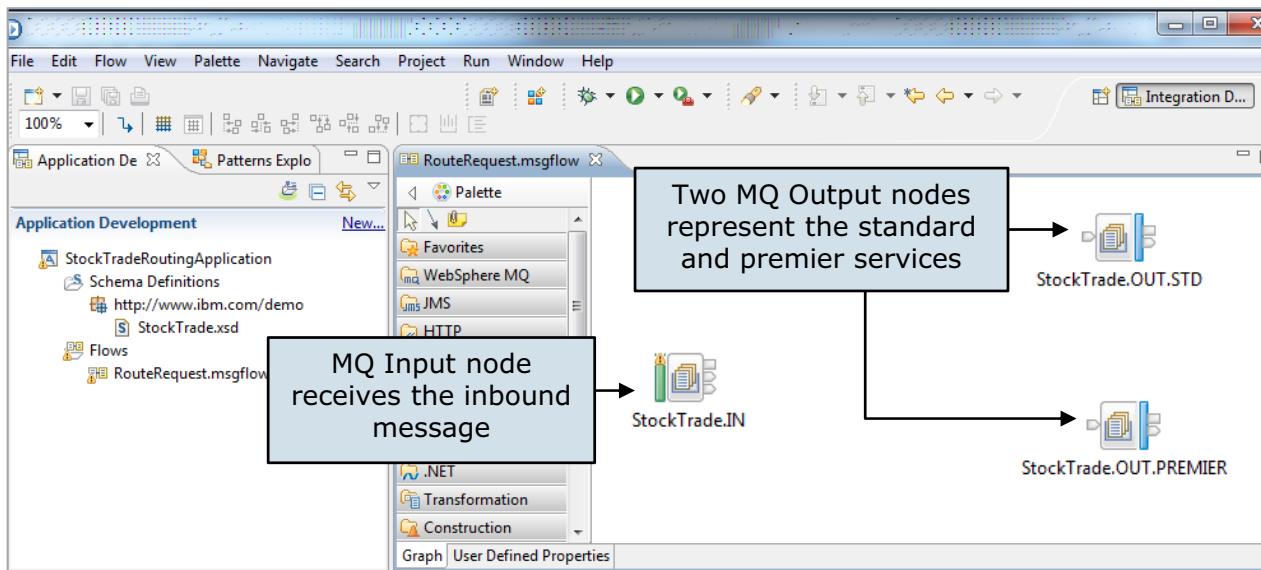
```
mqsimode -x DecisionServices
```

On z/OS, you must enable or disable decision services for each integration node by specifying the integration node name in the `mqsimode` command:

```
mqsimode IntNode -x DecisionServices
```



## Stock Trade example: Message flow



- Use a decision service to determine whether the “standard” or “premier” service is used

© Copyright IBM Corporation 2016

Figure 11-9. Stock Trade example: Message flow

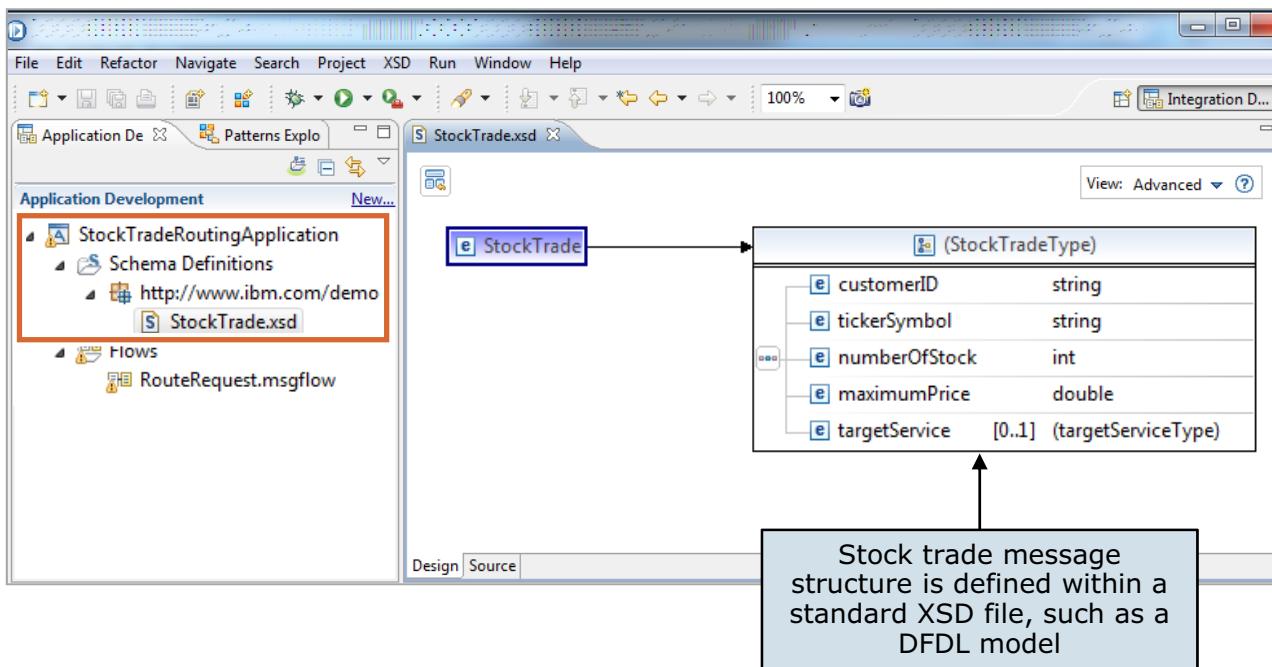
WM676/ZM6761.0

### Notes:

This slide and the next slides provide an example of how to create and implement a decision service. In this example, the decision service is added to a message flow that gets a message off an IBM MQ queue and puts a message to one of two output queues. The destination output queue is determined from the business rule that determines whether the “standard” or “premier” service is used for the message.

The example uses the Integration Toolkit Rules editor to create the business rule.

## Stock Trade example: Message model



© Copyright IBM Corporation 2016

Figure 11-10. Stock Trade example: Message model

WM676/ZM6761.0

### Notes:

A decision service requires a schema that describes the message data. The schema files can be imported into the message flow application or referenced in a shared or static library.

Required schema files must be in the scope of the decision service. If you create a decision service in an application, ensure that the required schema files are available in that application or in an associated library. You can use XMLNSC or DFDL schemas.

The schema in the figure describes a stock trade message. The stock trade message contains a customer ID, ticker symbol, number of shares of stock, maximum stock price, and target service.



## Stock Trade example: Define the decision service (1 of 2)

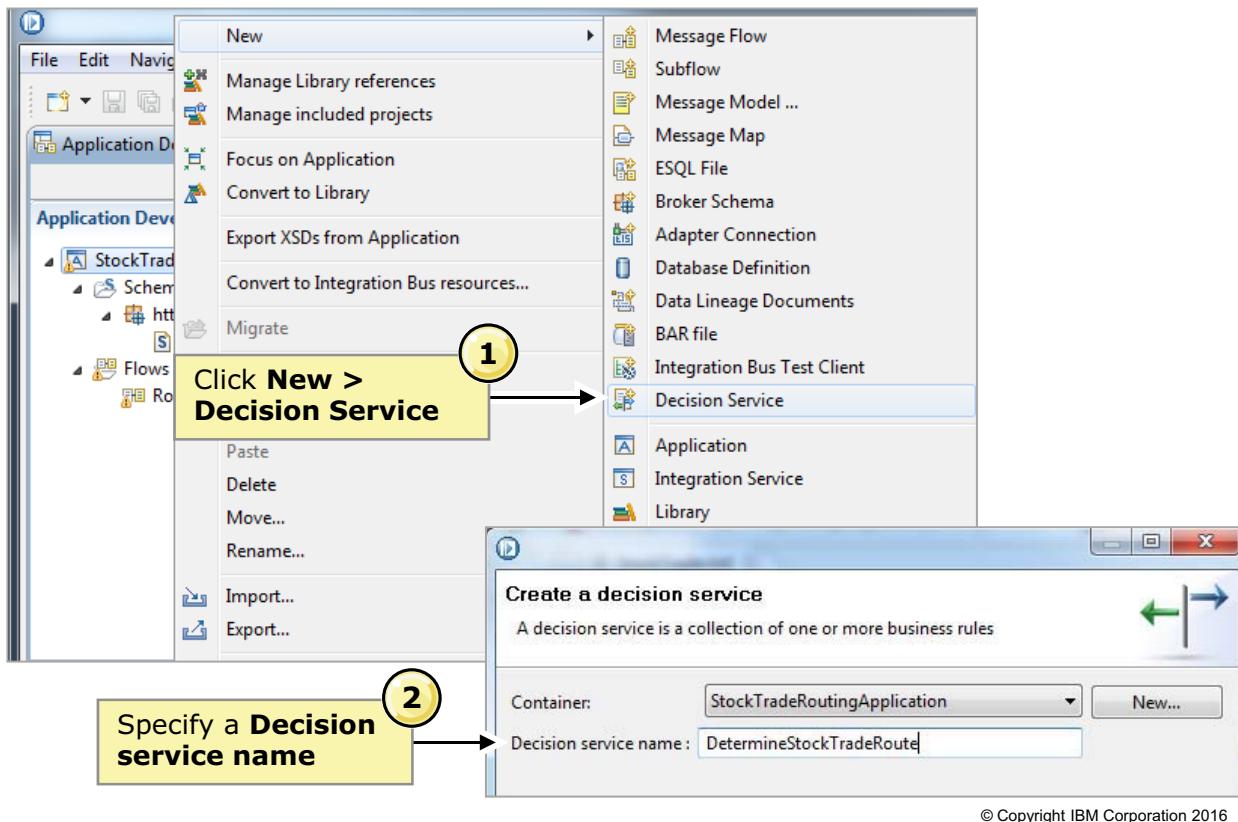


Figure 11-11. Stock Trade example: Define the decision service (1 of 2)

WM676/ZM6761.0

### Notes:

You can create a decision service in the Integration Toolkit by clicking **New > Decision Service**.

On the **Create a new decision service** window, select an existing container or create a new one. You can create a decision service in an application, library, or integration project. You also specify a name for the decision service on this window. The decision service name cannot contain spaces.

In this example, the decision service is created in an application that is named StockTradeRoutingApplication. The decision service is named DetermineStockTradeRoute.

## Stock Trade example: Define the decision service (2 of 2)

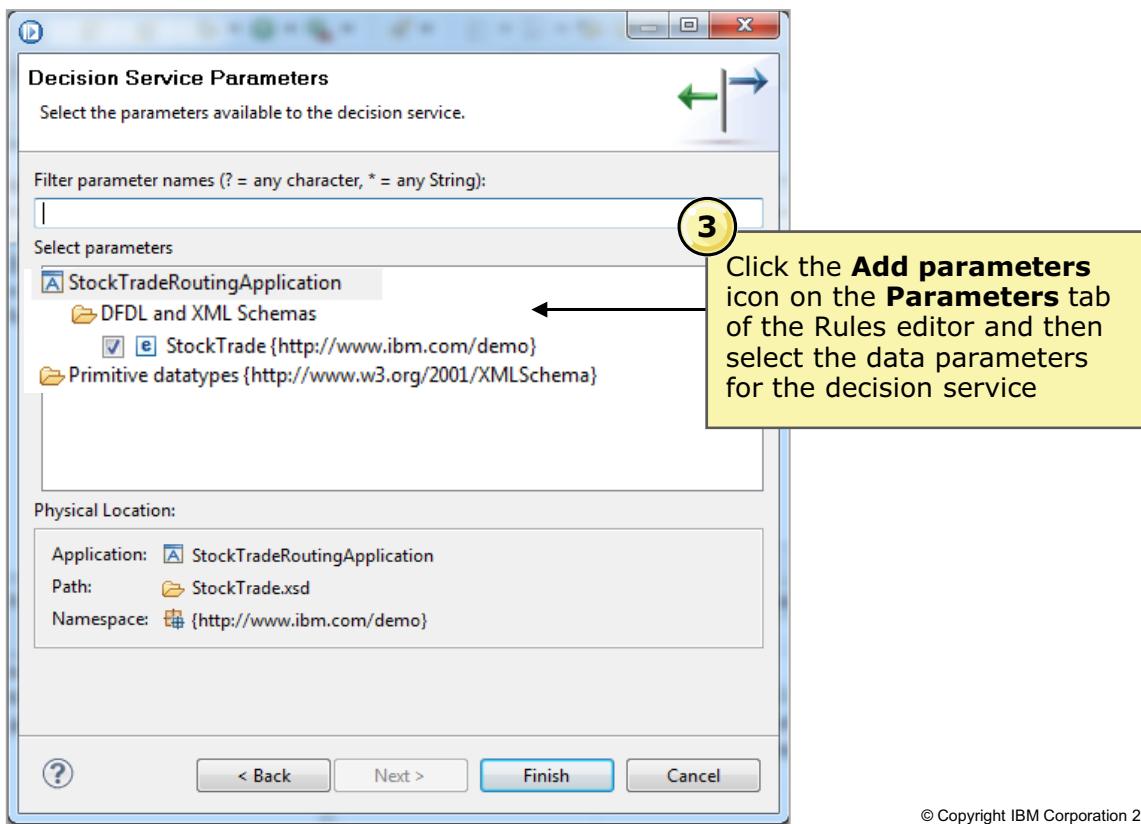


Figure 11-12. Stock Trade example: Define the decision service (2 of 2)

WM676/ZM6761.0

### Notes:

Decision service requires data parameters. The next step for defining a decision service is to select data parameters from a list of global elements, global types, and simple types.

The types of the selected items serve as data parameter types of the decision service.

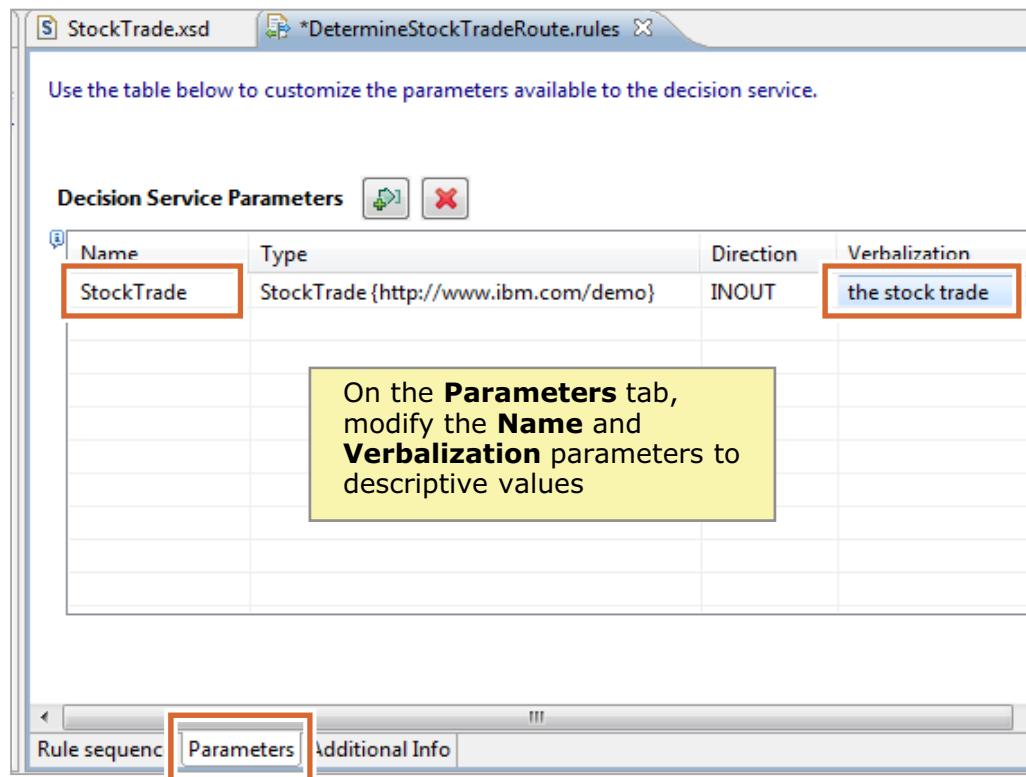
This list of parameters can be modified later in the Decision Service editor.

Required schema files must be in the scope of the decision service. If you create a decision service in an application, ensure that the required schema files are available in that application or in an associated library.

You can use XMLNSC or DFDL schemas.

WebSphere Education 

## Stock Trade example: Add decision service rules (1 of 2)



© Copyright IBM Corporation 2016

Figure 11-13. Stock Trade example: Add decision service rules (1 of 2)

WM676/ZM6761.0

### Notes:

Clicking **Finish** on the **Create Decision Service** wizard, open the Decision Service editor in the Integration Toolkit.

The first step in the Decision Service editor is to customize the parameters on the **Parameters** tab.

All the parameters of the decision service are listed in the **Decision Service Parameters** table. To edit a value in the table, click the table cell and then edit the value in the cell.

- The **Name** cell specifies a unique name for the parameter. You can override the value in the Name field with more descriptive names. In the example, the parameter name is **Stock Trade**.
- The **Type** cell identifies the selected item type.
- The **Direction** cell indicates the direction of the parameter (IN, OUT, INOUT). Currently, only INOUT is supported for decision services that are authored in the Integration Toolkit.
- The **Verbalization** cell specifies a natural language name (or alias) for the parameter. In the example, the **Verbalization** cell for the **Stock Trade** type is **the stock trade**. This value is used to refer to the parameter when writing a rule.

You can add one or more parameters by clicking the **Add parameters** icon.

## Stock Trade example: Add decision service rules (2 of 2)

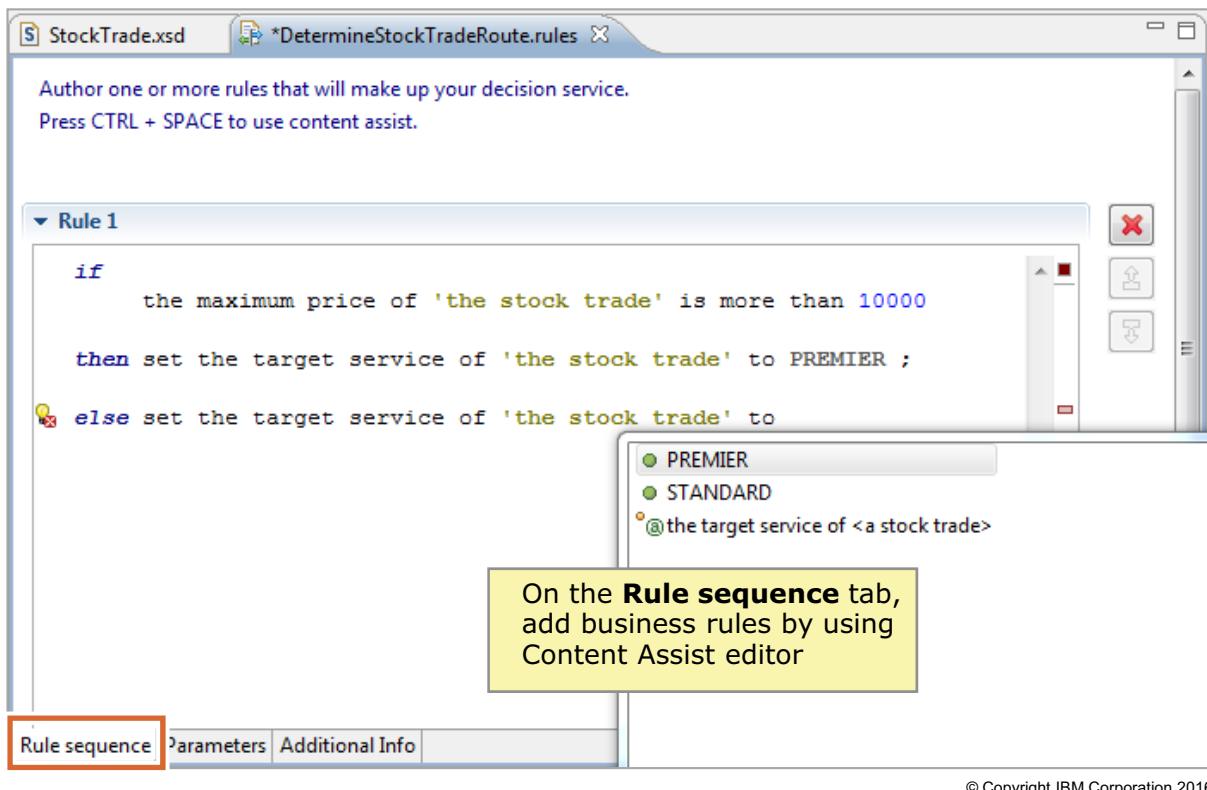


Figure 11-14. Stock Trade example: Add decision service rules (2 of 2)

WM676/ZM6761.0

### Notes:

In the Decision Service editor, use the **Rule sequence** tab to add rules and specify the order in which they are run.

You use the vocabulary that you created on the **Parameters** tab to write your rules. Initially, one empty rule is provided for you to populate.

A business rule typically consists of definitions, conditions, and actions. Use content assist to help construct your rules. To access content assist, press Ctrl+Space to display a list of available options. Content assist also opens automatically when you type a Space character in the rule. The options that you see are based on the schema parameters that you selected.

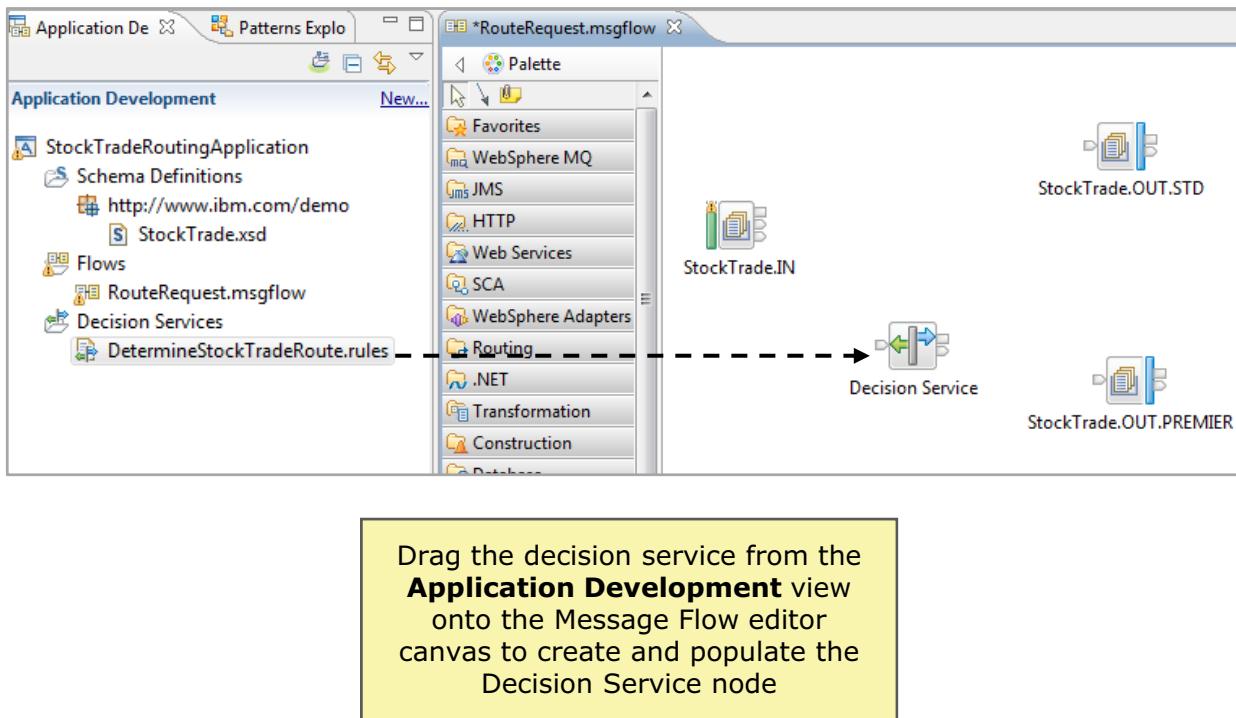
The business rule determines the route that the message takes through the message flow. The rule in this example is:

```
if
the maximum price of 'the stock trade' is more than 10000
then set the target service of 'the stock trade' to PREMIER ;
else set the target service of 'the stock trade' to STANDARD ;
```

A semicolon character is used to end each action statement.



## Stock Trade example: Embed the decision service



© Copyright IBM Corporation 2016

Figure 11-15. Stock Trade example: Embed the decision service

WM676/ZM6761.0

### Notes:

After you complete all the business rules on in the Decision Service editor, you can drag the decision service from the **Application Development** view onto the Message Flow editor. Dragging the decision service onto the Message Flow canvas creates and populates the Decision Service node based on the decision service configuration.

In the example, the `DetermineStockTradeRoute.rules` decision service is added to the `RoutingRequest.msgflow` message flow.

## Stock Trade example: Configure the Decision Service node

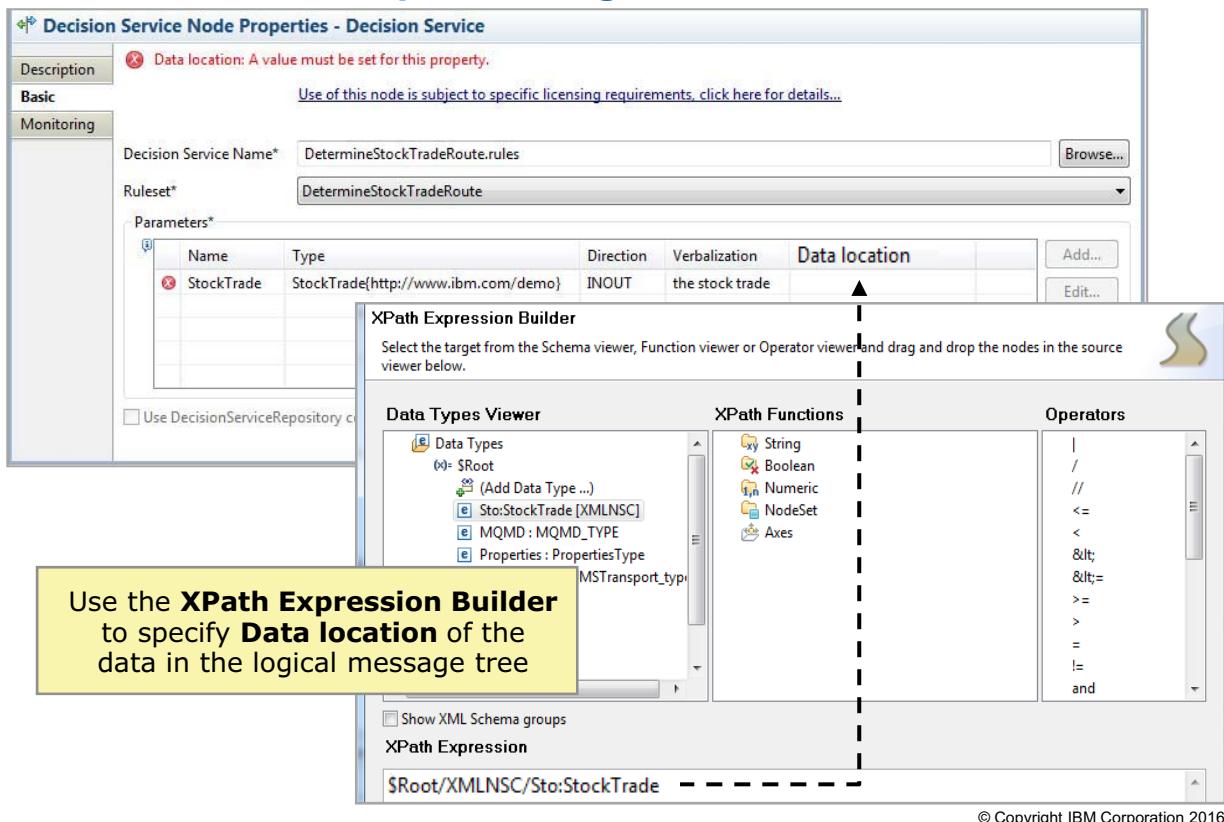


Figure 11-16. Stock Trade example: Configure the Decision Service node

WM676/ZM6761.0

### Notes:

Before you can deploy a message flow that contains a Decision Service node, you must specify the logical message tree location of the data that the parameter represents in the Decision Service node **Properties**.

In the example, you add a data location by clicking the **StockTrade** parameter and then clicking **Edit**. You can type the data location or click **Edit** again to access the XPath Expression Builder. In the example, the data location for the **StockTrade** parameter is `$Root/XMLNSC/Sto:StockTrade`.

## Stock Trade example: Complete the message flow

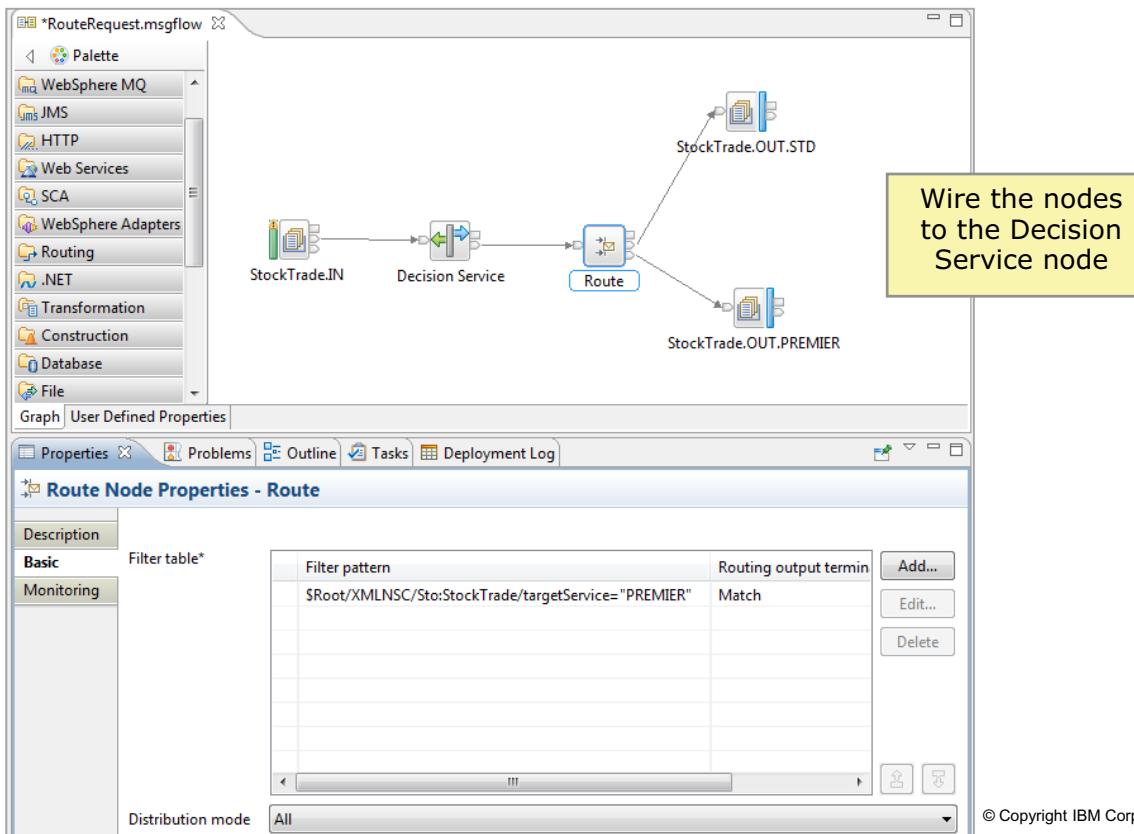


Figure 11-17. Stock Trade example: Complete the message flow

WM676/ZM6761.0

### Notes:

The final step in the Stock Trade example is to wire the Decision Service node to the MQ Input node and the Route node. In the example, the Route node routes the message based on the **TargetService** that the Decision Service node sets.

## Using a rule application archive

- Import the rule application archive
- If the decision service is created from a rule application archive that was imported from IBM Operational Decision Manager:
  - You cannot add, edit, or delete parameters
  - You cannot see or edit the rules that are contained in the rule set
- Configure the connection to the Operational Decision Manager Rules Execution Server database
  1. Create a JDBCProviders configurable service.
  2. Configure the integration node security identity for the database.

Example: `mqsisetdbparms IBNODE -n jdbc::administrator -u administrator -p passw0rd`

  3. Create a DecisionServiceRepository configurable service.
  4. Select the **Use DecisionServiceRepository configurable service** property on the Decision Service node.

© Copyright IBM Corporation 2016

Figure 11-18. Using a rule application archive

WM676/ZM6761.0

### Notes:

To use rules that are written in IBM Operational Decision Manager to process messages in Integration Bus, import a rule application archive into the Integration Toolkit. At run time, you can retrieve those business rules directly from an external IBM Operational Decision Manager repository. For example, you might use the local copy of the business rules for testing and then later switch to use the external repository copy.

After you import a rule application archive into Integration Bus, you can extract the schema files from it. You can use the extracted schema files to set parameters on the Decision Service node. You might also want to extract the schema files, for example, to create a wrapper schema for use with validation or mapping of the input message.

When you open an imported decision service in the Decision Service editor, you see only the **Parameters** and **Additional Info** tabs, and you cannot edit the information that is on these tabs.

You must define a JDBCProviders configurable service that defines the connection details to the database repository. You must define the following properties on the JDBCProviders configurable service to identify the database repository:

- **databaseName**: The data source name of the database repository
- **serverName**: The server on which the database repository is hosted
- **portNumber**: The port number on which the server is listening
- **securityIdentity**: The identity token that was used on the `mqsisetdbparms` command to store the user ID and password.

The **DecisionServiceRepository** configurable service identifies the connection to an external repository at run time. This configurable service references the JDBCProviders configurable service that defines the connection to the database repository. You can update the default **DecisionServiceRepository** configurable service only. You cannot create multiple **DecisionServiceRepository** configurable services.



## Importing a rule application archive

1. In Operational Decision Manager, export the RuleApp archive as a JAR file.
2. In the Integration Toolkit, click **Import > Decision Services > Rule application archive file** and then select the JAR file.
3. On the Decision Service node in the message flow:
  - a. Select the decision service
  - b. Specify the **Data location**
  - c. Select the **DecisionServiceRepository** configurable service property on the Decision Service node

© Copyright IBM Corporation 2016

Figure 11-19. Importing a rule application archive

WM676/ZM6761.0

### Notes:

If you create rules in IBM Operational Decision Manager, you can import them and use them in IBM Integration Bus. The resources that you import from IBM Operational Decision Manager must meet the following criteria:

- The imported file must be a rule application archive.
- The application rule sets in the rule application archive must contain action rules.
- Application rule sets must have a schema XML Object Model (XOM). (Java XOMs are not supported.)
- Application rule sets must have parameters that have schema types.

Export the rule application archive from the IBM Operational Decision Manager rule designer by using the **Rule Designer** menu or the **RuleApp** editor. You can also download a rule application archive from an IBM Operational Decision Manager rule execution server console.

The imported rule application archive is converted to a decision service (.rules file). The decision service opens in the Decision Service editor.




## DecisionServiceRepository configurable service

**Default - DecisionServiceRepository Configurable Service**

**Overview**

**Properties**

resManagementProfiling	false
resManagementPort	1883
JDBCProviderName	ODM85DB
resManagementHost	localhost

- Create a configurable service by using the IBM Integration web interface or an IBM Integration command
- **JDBCProviderName** must match the JDBCProvider configurable service that specifies connection to the Operational Decision Manager Rules Execution Server database
  - Verify that the Rule Execution Server is configured to publish rule changes over TCP/IP
- **resManagementPort** must match the management port that is specified for the Rule Execution Server

© Copyright IBM Corporation 2016

Figure 11-20. DecisionServiceRepository configurable service

WM676/ZM6761.0

### Notes:

You can modify and create configurable services by using the Integration web interface or the `mqsicreateconfigurableservice` command.

The **JDBCProviderName** property specifies the name of a JDBCProviders configurable service that defines the connection details to the database repository.

The **resManagementHost** property specifies the host on which the Rule Execution Server console is deployed. You specify this property with the **resManagementPort** property to register with the Rule Execution Server management console. If this Rule Execution Server management console is configured for TCP/IP notifications, the integration servers can receive update notifications when the contents of rule applications change.

For the DecisionServiceRepository configurable service, changes to the property values of this configurable service or the associated JDBCProviders configurable service take effect immediately. You do not need to restart the integration server.



## Monitoring a decision service

- Decision service resource statistics
  - Name of the decision service
  - Number of decisions that the decision service processes successfully
  - Number of decisions that the decision service does not process successfully
  - Cumulative total of rules that are triggered (matched) by the messages that the decision service processes
- Decision service activity log includes the following types of activity
  - Decision service ran successfully (indicates the number of rules that were matched)
  - Decision service failed to run successfully
  - Decision service repository that the configurable service defines was accessed for the first time

© Copyright IBM Corporation 2016

Figure 11-21. Monitoring a decision service

WM676/ZM6761.0

### Notes:

You can monitor decision services by using the Integration Bus resource statistics and the activity log.

The figure lists the monitoring information that is available for decision services.



## Unit summary

Having completed this unit, you should be able to:

- Create a decision service that implements business rules to provide routing, validation, and transformation

© Copyright IBM Corporation 2016

---

Figure 11-22. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or False: To use the IBM Decision Server component beyond development and functional test, you must purchase a separate license entitlement for either IBM Decision Server or IBM Decision Server Rules Edition for Integration Bus.

© Copyright IBM Corporation 2016

Figure 11-23. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answer here:

1.



## Checkpoint answers

1. True or False: To use the IBM Decision Server component beyond development and functional test, you must purchase a separate license entitlement for either IBM Decision Server or IBM Decision Server Rules Edition for Integration Bus.

Answer: **True**

© Copyright IBM Corporation 2016

Figure 11-24. Checkpoint answers

WM676/ZM6761.0

### Notes:



## Exercise 6



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 11-25. Exercise 6

WM676/ZM6761.0

### Notes:

In this exercise, you use the IBM Integration Toolkit to create a decision service, which is a collection of rules that are used to process a message. You then test the decision service by using it in a message flow.



## Exercise objectives

After completing this exercise, you should be able to:

- Create a decision service
- Write business rules in the IBM Integration Toolkit Rule Designer
- Test the decision service

© Copyright IBM Corporation 2016

---

Figure 11-26. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.

# Unit 12. Developing integration solutions by using a REST API

## What this unit is about

In IBM Integration Bus, a REST API is a specialized application that can be used to expose integrations as a RESTful web service that HTTP clients can call. Swagger is an open standard for defining a REST API. This unit describes how to create a REST API from a Swagger 2.0 document and implement a REST API operation.

## What you should be able to do

After completing this unit, you should be able to:

- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API

## How you will check your progress

- Checkpoints

## References

IBM Knowledge Center for IBM Integration Bus

## Unit objectives

After completing this unit, you should be able to:

- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API

© Copyright IBM Corporation 2016

---

Figure 12-1. Unit objectives

WM676/ZM6761.0

### Notes:

## REST API resources

- REST API describes a set of **resources** and a set of **operations** that can be called on those resources

- REST API **base path** is root from which all of the resources and operations are available

Example base path: `http://mycompany.com:7843/customerdb/v1`

- Each resource in a REST API has a path, relative to the base path that identifies that resource

Example resources:

```
/customers
/customers/12345
/customers/12345/orders
/customers/12345/orders/67890
```

All customers in the database  
Customer 12345  
All orders for customer 12345  
Order 67890 for customer 12345

© Copyright IBM Corporation 2016

Figure 12-2. REST API resources

WM676/ZM6761.0

### Notes:

A REST API describes a set of resources, and a set of operations that can be called on those resources.

The REST API has a base path, which is similar to a context root. All resources in a REST API are defined relative to its base path.

The base path can provide isolation between different REST APIs, and isolation between different versions of the same REST API. For example, a REST API can be built to expose a customer database over HTTP. The base path for the first version of that REST API might be `/customerdb/v1`, while the base path for the second version of that REST API might be `/customerdb/v2`.

The REST API describes a set of resources. The HTTP client uses a path relative to the base path that identifies the resource in the REST API that the client is accessing. The paths to a resource can be hierarchical, and a well-designed path structure can help a consumer of a REST API understand the resources available within that REST API.

## REST API operations

- Each resource in a REST API has a set of operations
- Each operation has a name and an HTTP method
- Operations can be called from any HTTP client
- Combination of the path of the HTTP request and the HTTP method identifies which resource and operation is being called

Example operations on the resource /customers/12345:

getCustomer

Call with HTTP GET to retrieve the customer details

updateCustomer

Call with HTTP POST to update the customer details

deleteCustomer

Call with HTTP DELETE to delete the customer

© Copyright IBM Corporation 2016

Figure 12-3. REST API operations

WM676/ZM6761.0

### Notes:

The operations in a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser.

Each resource in the REST API has a set of operations that an HTTP client can call. An operation in a REST API has a name and an HTTP method (such as GET, POST, or DELETE).

The name of the operation must be unique across all of the resources in that REST API. Also, a single resource can have only one operation that is defined for a specific HTTP method. The combination of the path and the HTTP method that the HTTP client that is making the request specifies is used to identify the resource and operation that is being called.

## REST API parameters

- Each REST API operation can specify a set of parameters
  - Parameters can be used to pass information to the operation
  - Parameters are in addition to the body passed in the HTTP request
- Integration Bus supports three types of parameters
  - **Path parameters:** One or more parts of the path for a resource can be defined as a variable
 

Example: The customer ID in the path parameter

```
/customers/{customerId}/orders/{orderId}
```

```
/customers/12345/orders/56789
```
  - **Query parameters:** One or more parameters can be specified in the URL following the path
 

Example: /customers?min=5&max=20
  - **Header parameters:** One or more parameters can be specified in the headers of the HTTP request

© Copyright IBM Corporation 2016

Figure 12-4. REST API parameters

WM676/ZM6761.0

### Notes:

Each operation in a REST API can have a set of parameters that an HTTP client can use to pass arguments into the operation. Each parameter must be defined in the definitions for the REST API. REST APIs in Integration Bus support several types.

- **Path parameters** can be used to identify a particular resource. The value of the parameter is passed to the operation by the HTTP client as a variable part of the URL. The value of the parameter is extracted from the path for use in the operation. Path parameters are denoted by using the syntax {paramName} in the path to the resource. For example, the customer ID can be passed in as a path parameter named: customerId: /customers/{customerId}
- The value of a **query parameter** is passed to the operation by the HTTP client as a key value pair in the query string at the end of the URL. For example, query parameters can be used to pass in a minimum and maximum number of results that a particular operation should return: /customers?min=5&max=20
- The HTTP client can pass **header parameters** to an operation by adding them as HTTP headers in the HTTP request. For example, header parameters might be used to pass a unique identifier that identifies the HTTP client that is calling the operation.

## Swagger

- Swagger is an open standard for defining a REST API
- Swagger document includes definitions of the resources, operations, and parameters in a REST API
  - Swagger document can also include a JSON schema that describes the structure of the request and response bodies to an operation
- Swagger open source tools can be used to interact with Swagger documents and the REST APIs that they describe
  - Swagger Editor can be used to help the development of a Swagger document
  - Swagger Editor can be downloaded and run locally, or is also hosted on the web

© Copyright IBM Corporation 2016

Figure 12-5. Swagger

WM676/ZM6761.0

### Notes:

Swagger is an open standard for defining a REST API. A Swagger document is the REST API equivalent of a WSDL document for a SOAP web service.

The specification for Swagger 2.0 can be found at:

<https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md>

## Integration Bus support for Swagger documents

- Integration Bus supports Swagger 2.0 JSON document
- To build a REST API in Integration Bus, you must develop and supply a Swagger 2.0 document that describes the REST API
- After a Swagger document is built in the Swagger Editor, click **File > Download JSON**
- To learn more:
  - Sample Swagger document `swagger.json` is provided in the `<IntegrationBus_install_dir>/server/sample/restapis/` directory
  - IBM Integration Toolkit tutorial: **Manage a set of records with IBM Integration Bus REST API services**

© Copyright IBM Corporation 2016

Figure 12-6. Integration Bus support for Swagger documents

WM676/ZM6761.0

### Notes:

To create a REST API in IBM Integration Bus, you must first define the resources and operations that are available in the new REST API. Those definitions must be specified in a Swagger document.

IBM Integration Bus places certain restrictions on the format of Swagger documents that are used to create REST APIs:

- The Swagger document must be saved as a `.json` file.
- You must enter a value for the `operationId` field in the operation object, and the value of the `operationId` field must be unique across all operations that are defined in this REST API. IBM Integration Bus uses the operation ID as a unique name to refer to the operation.
- The base paths of all REST APIs deployed to a single integration server must be unique. You cannot deploy a REST API to an integration server if another REST API is already deployed with the same base path.
- Form data parameters are allowed in Swagger documents by IBM Integration Bus, but no support is provided for parsing form data parameters.

## Creating a REST API in Integration Bus

- With an Integration Bus REST API, you can expose a set of integrations as a RESTful web service
  - Support for all of the Integration Bus features that you can use with applications (such as shared libraries, monitoring, and activity logs)
  - Support for all message flow nodes
- Operations that are defined in the REST API are implemented as subflows
  - REST API container automatically handles the routing of inbound HTTP requests to the correct subflow for the operation that is called
  - To implement, connect the Input and Output nodes in each subflow
- Requires a Swagger document
  - Can be imported by using the **Create a REST API** wizard

© Copyright IBM Corporation 2016

Figure 12-7. Creating a REST API in Integration Bus

WM676/ZM6761.0

### Notes:

In earlier versions of Integration Bus, implementing a REST API required the developer to handle inbound HTTP requests, and routing of those requests. Programming was required to process variable paths in the request, and extract path parameters.

To create an integration solution that uses a REST API, complete the following steps:

1. Create a Swagger document that describes the resources and operations that are available in the REST API.
2. Create a REST API in the Integration Toolkit by importing the Swagger document.
3. Implement each of the operations in the REST API as a subflow. See “Implementing an operation in a REST API.”

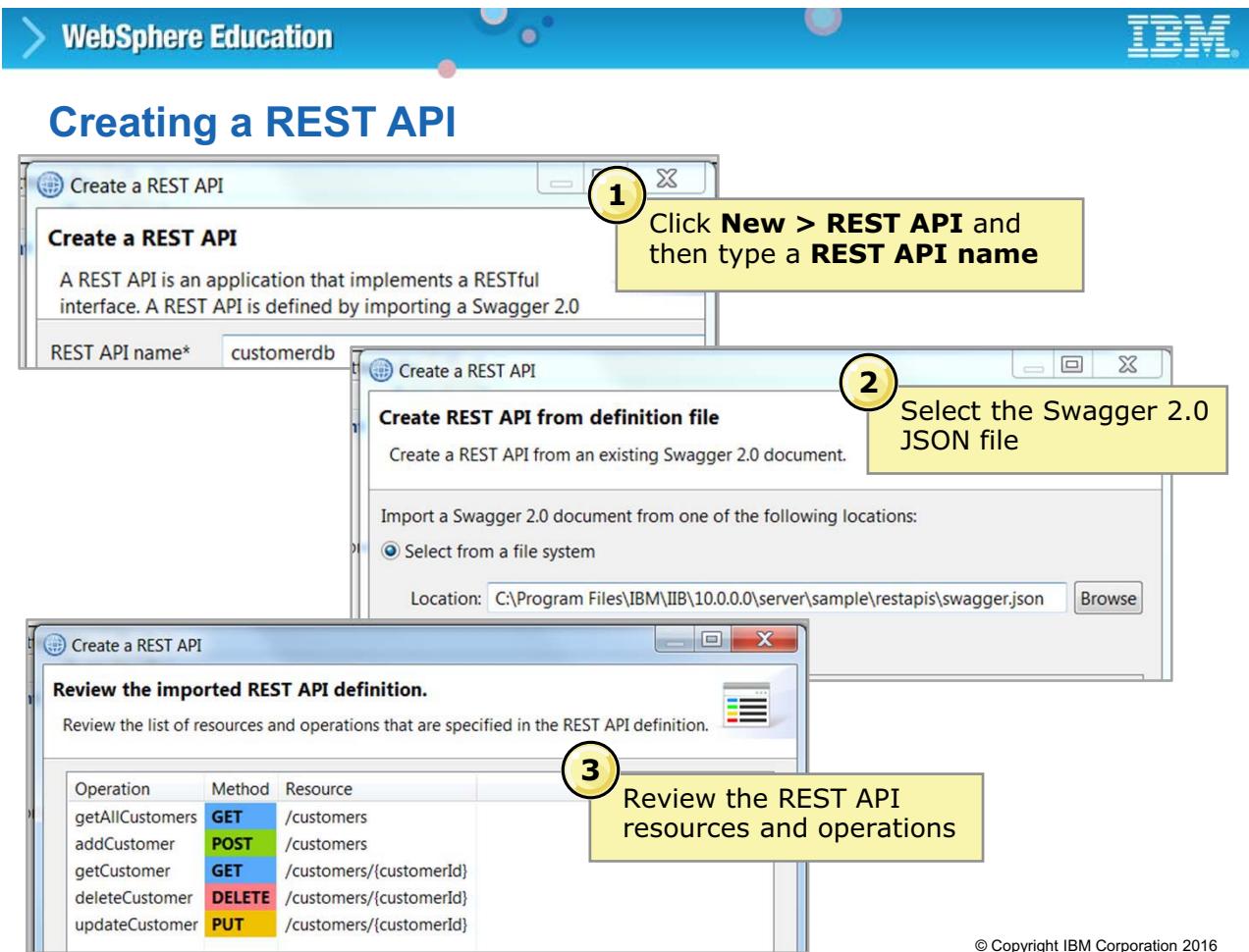


Figure 12-8. Creating a REST API

WM676/ZM6761.0

## Notes:

To create a REST API in IBM Integration Bus, you must import a Swagger document by using the Integration Toolkit.

To create a REST API by importing a Swagger document, complete the following steps:

1. Open the **Create a REST API** wizard by clicking **File > New > REST API**.
2. Enter a name for the REST API. The name that you specify is used as the name of the project in the IBM Integration Toolkit, and the name of the REST API when it is deployed to an integration server.
3. Review the REST API resources and operations.

## REST API Description view

The screenshot shows the REST API Description view for a database named 'customerdb'. It includes a header with the base URL: /customererdb/v1. The main content area is divided into sections:

- List of resources and operations**: A callout box points to the operations section.
- Operations**: Shows two operations: 'getAllCustomers' (GET) and 'addCustomer' (POST). The 'getAllCustomers' row has a yellow box labeled 'Click to implement the operation' and a red box around the 'Implement the operation' link.
- /customers/{customerId}**: A yellow box labeled 'Click to implement error handling' covers the error handling links below.
- Error Handling**: Three links are listed: 'Implement the Catch handler', 'Implement the Failure handler', and 'Implement the Timeout handler'. The first two are in a red box.

- Click **Implement the operation** to generate a subflow for the operation and create the links between the REST API and the subflow
- Click the **Error Handling** links to implement error handling for errors and exceptions that the subflow does not handle for an operation

© Copyright IBM Corporation 2016

Figure 12-9. REST API Description view

WM676/ZM6761.0

### Notes:

The REST API **Description** view is the starting point for a newly created REST API.

From the REST API **Description** view, you can see the list of the resources and operations that are defined in the REST API. You can also see any parameters that are defined for those operations.

You must use the REST API **Description** view to implement an operation in a REST API. Clicking an unimplemented operation automatically generates a subflow for that operation, and creates the underlying links between the REST API and the subflow.

It is not an error to implement all operations in a REST API before deploying it to an integration server. You can implement an operation, deploy the REST API, test the new operation, and then repeat. Unimplemented operations return an `HTTP 501 Not Implemented` status code when called by an HTTP client.

You can also use the REST API **Description** view to implement error handling for a REST API. Error handlers are available to handle errors and exceptions that the subflow for an operation does not handle.

Finally, you can also enable HTTPS from the REST API **Description** view.

## Implementing an operation

- Click **Implement the operation** in the **Description** view to generate an empty subflow
- When the operation is called by an HTTP client, a message is passed to the subflow Input node for that operation
  - If a body is provided in the request, the message has a JSON request body
  - JSON is the default message domain, but you can use other message domains
- When the subflow completes and passes a message to the Output node, the response is passed back to the HTTP client
- Any parameters (path, query, and header) defined by the operation are extracted from the HTTP request and stored in the LocalEnvironment tree
  - Access the extracted parameter values in the LocalEnvironment tree from message flow nodes by using the Compute, Java Compute, and .NET Compute nodes
  - Access the extracted parameter values by using an Integration Bus graphical data map to map the LocalEnvironment tree

© Copyright IBM Corporation 2016

Figure 12-10. Implementing an operation

WM676/ZM6761.0

### Notes:

Operations in a REST API are implemented as a subflow. You must use the REST API Description to create an empty subflow for each operation. You can then implement the operation by adding any of the standard message flow nodes that are available with IBM Integration Bus to the subflow.

To implement an operation in a REST API:

1. Open the REST API Description for the REST API that contains the operation that you want to implement. The REST API Description is in the **Application Development** view under the REST API project.
2. Locate the operation in the REST API **Description**. Operations are listed under the **Operations** heading, and are grouped according to resource.
3. Click the **Implement the operation** link. A new subflow is automatically created and opened.

## Accessing REST parameters: Programming examples

Compute node (ESQL)

```
DECLARE max INTEGER -1;
IF FIELDTYPE(InputLocalEnvironment.REST.Input.Parameters.max) IS NOT NULL
THEN
    SET max = InputLocalEnvironment.REST.Input.Parameters.max;
END IF;
```

Java Compute node

```
MbElement maxElement =
inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/Input/
Parameters/max");
int max = -1;
if (maxElement != null) {
    max = Integer.valueOf(maxElement.getValueAsString());
}
```

.NET Compute node

```
NBElement maxElement =
inLocalEnvironment.RootElement["REST"]["Input"]["Parameters"]["max"];
int max = -1;
if (max != null){
    max = (int) maxElement;
}
```

© Copyright IBM Corporation 2016

Figure 12-11. Accessing REST parameters: Programming examples

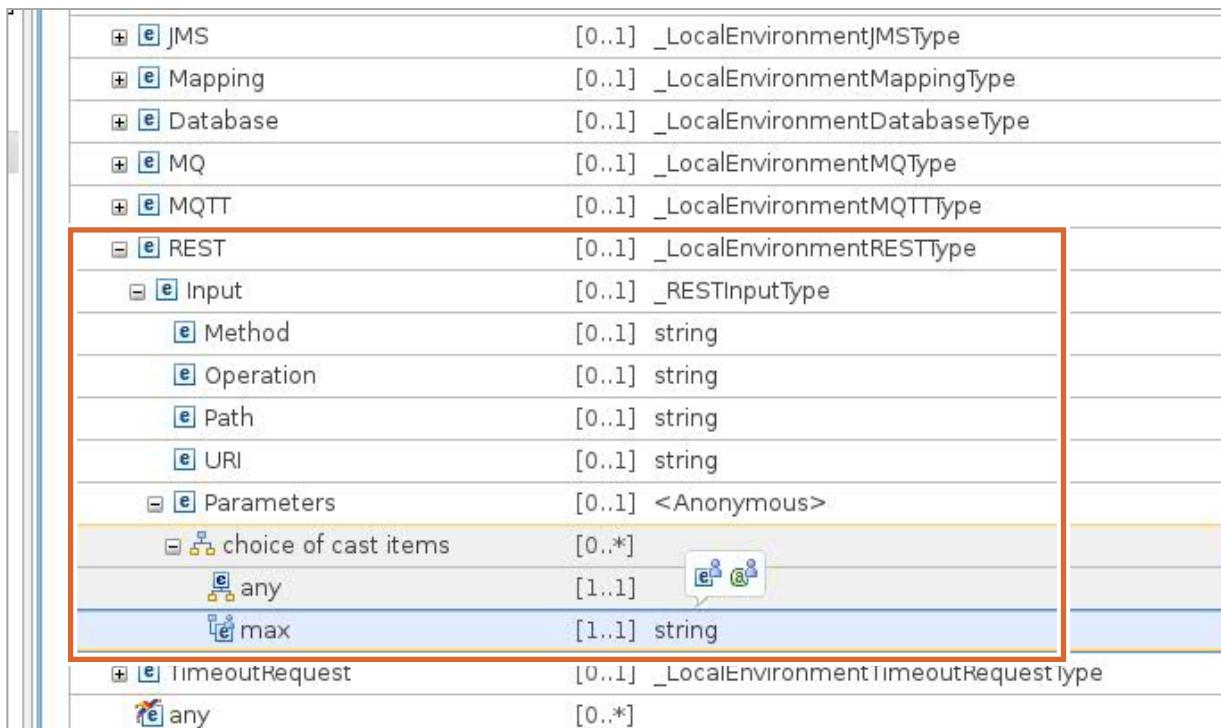
WM676/ZM6761.0

### Notes:

Information about the current operation is automatically placed into the LocalEnvironment tree. You can use this information in your implementation if you want to determine which operation in the REST API was called, which HTTP method was used, the request path, or the request URI.

The figure shows examples of accessing the LocalEnvironment tree by using ESQL (in a Compute node), Java (in a Java Compute node), and .NET (in a .NET Compute node).

## Accessing REST parameters in a graphical data map



© Copyright IBM Corporation 2016

Figure 12-12. Accessing REST parameters in a graphical data map

WM676/ZM6761.0

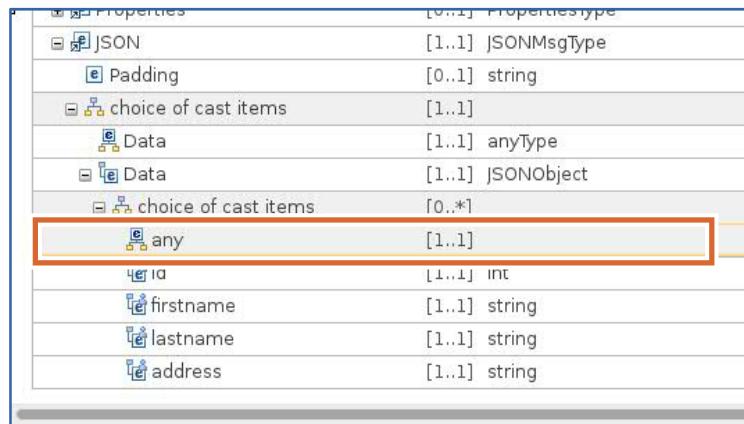
### Notes:

You can also access the REST parameters in the LocalEnvironment tree by using an Integration Toolkit graphical data map.

The current operation information is available as a set of elements, which are named **Method**, **Operation**, **Path**, and **URI**, in a LocalEnvironment message assembly under the **Local Environment > REST > Input** folder.

## Mapping JSON request bodies

- JSON mapping support in Integration Bus V10 can be used to map JSON request bodies in a REST API



- As of Integration Bus V10.0.0.0, there is no support for extracting and using the JSON schema information in the Swagger document in a graphical data map
  - Open source Swagger tools can use JSON schema information so you should define the request and response bodies in Swagger

© Copyright IBM Corporation 2016

Figure 12-13. Mapping JSON request bodies

WM676/ZM6761.0

### Notes:

Depending on the HTTP method of the operation, the operation can accept data from the HTTP client in the request body. REST APIs in IBM Integration Bus are configured by default to process JSON data.

You can also use a message map to process JSON data.



## Packaging and deploying REST APIs

- REST APIs can be packaged in a BAR file and deployed to an integration server by using any of the standard mechanisms (Integration Toolkit, Integration Bus commands, or the Integration API)
- After it is deployed, a REST API appears in the Integration Toolkit **Integration Nodes** view and IBM Integration web interface under **REST APIs**
- As of Integration Bus V10.0.0.0, REST APIs cannot be used with the integration node HTTP listener
  - If the integration node has a default queue manager, explicitly enable the HTTP listener for the integration server
- Can use the REST API base path to:
  - Isolate a REST API from other REST APIs
  - Isolate multiple versions of the same REST API on a single integration server
- You cannot deploy a REST API that would clash with URLs that other REST APIs deployed to an integration server handle, or HTTP Input nodes that are deployed outside of REST APIs

© Copyright IBM Corporation 2016

Figure 12-14. Packaging and deploying REST APIs

WM676/ZM6761.0

### Notes:

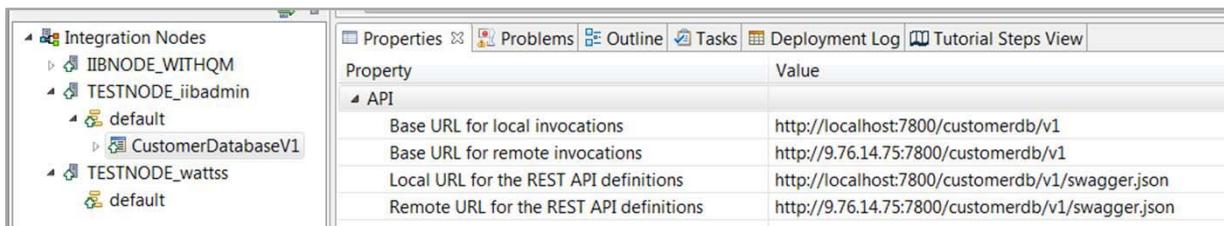
REST APIs can be packaged into a BAR file and deployed to an integration server by using any of the standard mechanisms such as the Integration Toolkit, Integration Bus commands, or the Integration API.

REST APIs can be deployed to integration servers that are configured to use the integration server HTTP listener only. REST APIs cannot be deployed to integration servers that are configured to use the integration node HTTP listener. If the integration node does not have a default queue manager, the integration server defaults to using the integration server HTTP listener.

If you plan to deploy more than one REST API to an integration server, the REST APIs must have different base paths.

## Finding the base URL of the REST API

1. Deploy the application.
2. Select the application in the **Integration Nodes** view.
3. View the API URLs in the **Properties** view.



- Use **Base URL for local invocations** or **Base URL for remote invocations** with HTTP client to call an operation in the deployed REST API
- **Local URL for the REST API definitions** and **Remote URL for the REST API definitions** is a URL that you can use to access the Swagger document for the deployed REST API

© Copyright IBM Corporation 2016

Figure 12-15. Finding the base URL of the REST API

WM676/ZM6761.0

### Notes:

## Deployed Swagger documents

- When a REST API is deployed, the Swagger document for that REST API is made available over HTTP from the same server and port that the REST API is hosted in
  - URL for the deployed Swagger document is available from the **Integration Nodes** view in the Integration Toolkit and in the Integration web interface
- Deployed Swagger document is automatically updated to reflect the server, port, and HTTP/HTTPS details for the deployed REST API
- You can use the deployed Swagger document with open source Swagger tools to explore and interact with a deployed REST API
- Pass the URL of the deployed Swagger document to a Swagger user interface to explore and test a deployed REST API
  - You must enable Cross-Origin Resource Sharing

© Copyright IBM Corporation 2016

Figure 12-16. Deployed Swagger documents

WM676/ZM6761.0

### Notes:

## Cross-Origin Resource Sharing (CORS)

- Origin of a web page is the scheme, host, and port of the web server that is hosting the web page
  - Same-origin request is made when a web page makes a request for another web page or resource on the same origin
  - Cross-origin request is made when a web page makes a request for another web page or resource on a different origin
- When making an HTTP request from a web page to a REST API or other HTTP service that is deployed to Integration Bus, it is likely that a cross-origin request must be made
  - Integration Bus includes built-in support for CORS on an integration server HTTP listener
  - Not enabled by default
  - Use the `mqsichangeproperties` command to enable:

```
mqsichangeproperties IntNode -e IntServer -o HTTPConnector
-n corsEnabled -v true
```
- Default settings for CORS should be sufficient for most needs, but more configuration options are available for fine level control
  - Changes are effective immediately

© Copyright IBM Corporation 2016

Figure 12-17. Cross-Origin Resource Sharing (CORS)

WM676/ZM6761.0

### Notes:

A web page has an origin. The origin of a web page is the scheme, host, and port of the web server that is hosting the web page. For example, the origin of <https://localhost:8080/test/index.html> is <https://localhost:8080>.

You can allow a web browser to access a REST API by using CORS. When you enable CORS on an integration server, it is enabled for all REST APIs and any other HTTP services that are running on that integration server. You are not required to configure CORS for each REST API that you deploy.

- Configure the integration server HTTP listener to enable CORS. You use the `mqsichangeproperties` command to enable CORS. The `connectorName` parameter for the HTTP listener is `HTTPConnector`, and for the HTTPS listener is `HTTPSSConnector`. If both HTTP and HTTPS are in use, you must configure the HTTP listeners for HTTP and HTTPS independently.
- Ensure that the CORS configuration meets the requirements for all operations that are deployed in the REST API. To allow cross-origin requests for more HTTP methods, other HTTP headers, or to allow authentication information to be passed into the REST API, you might have to change some extra parameters



## REST API administration

- All administrative and operational controls that are available for applications in Integration Bus are also available for REST APIs
- Integration web interface provides information about the resources, operations, and parameters that are available in a deployed REST API

**API tab contains the details of resources and operations within the deployed REST API**

© Copyright IBM Corporation 2016

Figure 12-18. REST API administration

WM676/ZM6761.0

### Notes:

After it is deployed, a REST API appears in the Integration Toolkit and web administration interface as a REST API, under the **REST APIs** category.

To view information about the deployed REST API in the web user interface, complete the following steps:

1. Expand the navigation of the deployed REST API. The currently deployed message flow, subflows, and any other deployed resources are displayed.
2. Click the down arrow for the deployed REST API to access the list of actions that are available for a deployed REST API. Depending on your access rights, you can start, stop, or delete the REST API, or start or stop statistics for the REST API.

In the **Quick View** section of the **Overview** tab, you can view the values for Base URL for local invocations and Base URL for remote invocations. By using an HTTP client, you can use one of these URLs, along with other details of the operation that is being called, to call an operation in the deployed REST API.

The **API** tab contains the details of resources, and operations within the deployed REST API are displayed. Operations are displayed with their HTTP method, name, description, and information about whether that operation is implemented as a subflow.

When you click an operation, the list of parameters for that operation are displayed, with their name, type, description, and whether a parameter is required.



## Unit summary

Having completed this unit, you should be able to:

- Create a REST API from a Swagger 2.0 document
- Implement a REST API operation
- Package and deploy a REST API

© Copyright IBM Corporation 2016

Figure 12-19. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. Complete the sentence: Operations that are defined in the REST API are implemented as \_\_\_\_\_ .
  - A. Subflows
  - B. Message flows
  - C. Integration services
2. In Integration Bus V10, REST APIs cannot be used with the integration server HTTP listener.

© Copyright IBM Corporation 2016

---

Figure 12-20. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.



## Checkpoint answers

1. Complete the sentence: Operations that are defined in the REST API are implemented as \_\_\_\_\_
  - A. Subflows
  - B. Message flows
  - C. Integration services

Answer: A
2. In Integration Bus V10, REST APIs cannot be used with the integration server HTTP listener.  
Answer: **False. In Integration Bus V10, REST APIs cannot be used with integration node HTTP listener. They can be used with the integration server HTTP listener.**

© Copyright IBM Corporation 2016

Figure 12-21. Checkpoint answers

WM676/ZM6761.0

### Notes:



# Unit 13. Using the global cache

## What this unit is about

A global cache is a repository for data that you want to reuse across processes (both in the same integration node, and across integration nodes). IBM Integration Bus supports an embedded global cache and you can configure IBM Integration Bus to connect to an external WebSphere eXtreme Scale grid. In this unit, you learn how to configure and use the IBM Integration Bus embedded global cache to store the data that you want to reuse.

## What you should be able to do

After completing this unit, you should be able to:

- Use the IBM Integration Bus global cache to store data that you want to reuse and share between integration nodes
- Configure properties of the embedded global cache by using commands or an XML cache policy file

## How you will check your progress

Checkpoints

## References

IBM Knowledge Center for IBM Integration Bus V10



## Unit objectives

After completing this unit, you should be able to:

- Use the IBM Integration Bus global cache to store data that you want to reuse and share between integration nodes
- Configure properties of the embedded global cache by using commands or an XML cache policy file

© Copyright IBM Corporation 2016

---

Figure 13-1. Unit objectives

WM676/ZM6761.0

### Notes:

## IBM Integration Bus global cache

- Stores state for integrations and static data in data structure that maps keys to values
  - Integration Bus internal cache
  - IBM WebSphere eXtreme Scale
- Must be explicitly enabled
- Default cache policy for Integration Bus internal cache is one cache across one integration node
- Alternatives to a cache policy
  - Control the topology by setting cache properties on the integration servers
  - Use an XML policy file to enable the cache across multiple integration nodes
- In Integration Bus Version 10.0.0.2 or later, the embedded global cache can use a WebSphere eXtreme Scale enterprise data grid
- Developer creates and accesses global cache data structures by using a Mapping node and JavaCompute node

© Copyright IBM Corporation 2016

Figure 13-2. IBM Integration Bus global cache

WM676/ZM6761.0

### Notes:

Integration Bus contains a global cache that you can use to store data that you want to reuse. The cache is not enabled by default. To use the cache, select an appropriate cache policy. The default cache policy creates a default topology of cache components in a single integration node. The alternatives to the default topology are: to have no policy, so that you can control your own topology by setting cache properties on the integration servers, or to use an XML policy file to enable the cache across multiple integration nodes.

Data in the global cache is stored in *maps*. A map is a data structure that maps keys to values. You can create and access global cache maps by using a JavaCompute and Mapping node.



**Note**

You can access the Java methods for global caching in a Compute node by using the ESQL CREATE PROCEDURE statement. For more information and examples, see the IBM DeveloperWorks blog article “How to configure the global cache for ESQL” in the IBM Integration Community: <https://developer.ibm.com/integration/>

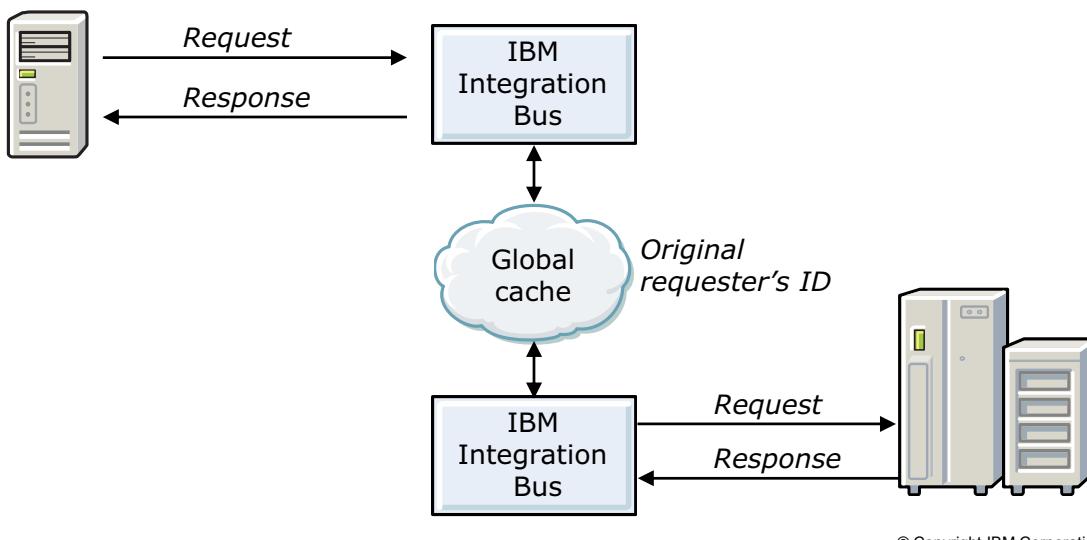
For example:

```
CREATE PROCEDURE readCache( IN P1 CHARACTER, IN P2 CHARACTER )
RETURNS CHARACTER
LANGUAGE JAVA
EXTERNAL NAME "com.ibm.broker.esql.GlobalCache.readCache";
```

You can also use a global cache in a WebSphere eXtreme Scale grid. In IBM Integration Bus Version 10.0.0.2 or later, the embedded global cache can use a WebSphere eXtreme Scale enterprise data grid. Enterprise data grids use the eXtreme data format (XDF) instead of Java object serialization. XDF allows for class evolution. With class evolution, you can evolve the class definitions that are used in the data grid without affecting older applications that are using previous versions of the class. For more information, see the “Enterprise data grid overview” topic in the WebSphere eXtreme Scale product documentation.

## Scenario 1: Storing state for integrations

- When Integration Bus integrates asynchronous systems, the integration node records information about the requester to correlate the replies correctly
- With a global cache, each integration node can handle replies, even when another integration node processes the request



© Copyright IBM Corporation 2016

Figure 13-3. Scenario 1: Storing state for integrations

WM676/ZM6761.0

### Notes:

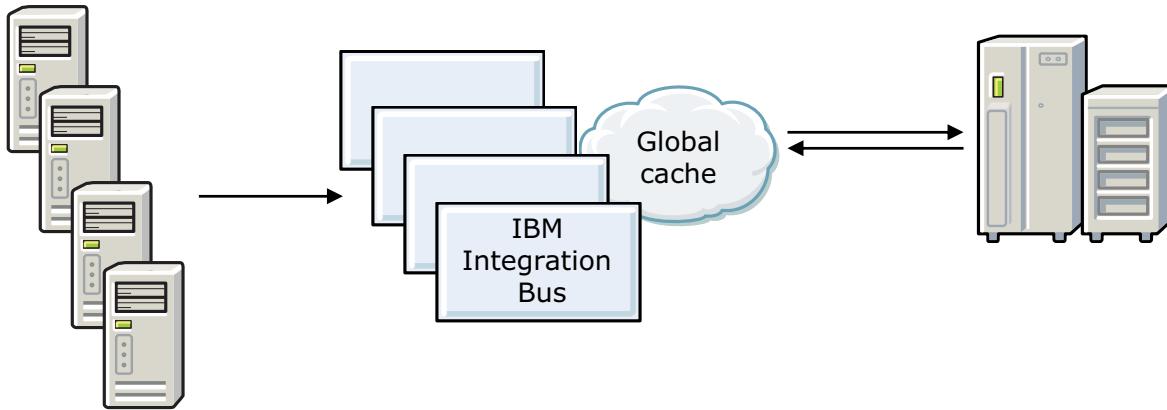
You might want to use a global cache to store information about a requester in a request/reply scenario.

For example, in an IBM MQ request/reply scenario, you typically need to keep the reply-to queue and original message ID in the MQMD. If the request goes through the network to another integration node, then it cannot restore the original reply-to queue and message ID, unless the originating integration node shared that information.

Before the implementation of a global cache, developers used a database or a queue to store the original requester's information. A global cache allows integration nodes to share this information in memory.

## Scenario 2: Caching infrequently changing data

- Use of a global cache facilitates horizontal scaling in situations where a cache is used to minimize network interactions to a back-end system
  - Increase the number of clients while maintaining a predictable response time for each client



© Copyright IBM Corporation 2016

Figure 13-4. Scenario 2: Caching infrequently changing data

WM676/ZM6761.0

### Notes:

You can also use a global cache to store cross-reference information to minimize network interactions with a back-end system.



## Global cache components

- Container server
  - Component that is embedded in the integration server that holds a subset of the cache data
  - All container servers in the global cache host all of the cache data at least one time
- Catalog servers
  - Controls the placement of data and monitors the health of containers
  - Must have at least one catalog server in the global cache
  - To avoid losing cache data when a catalog server is lost, use a policy file to specify more than one catalog server for an integration node

© Copyright IBM Corporation 2016

Figure 13-5. Global cache components

WM676/ZM6761.0

### Notes:

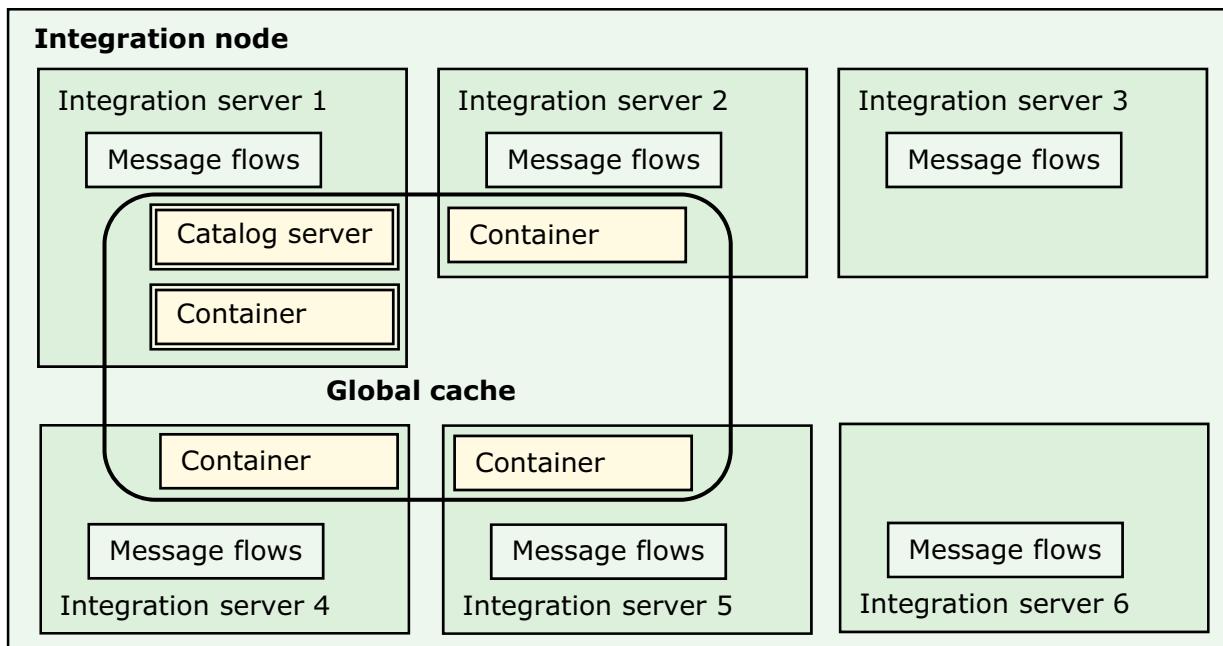
The primary components of the global cache are the container server and the catalog server.

A container server is a component that is embedded in the integration server that holds a subset of the cache data. Between them, all container servers in the global cache host all of the cache data at least one time. If more than one container exists, the default cache policy ensures that all data is replicated at least one time. In this way, the global cache can cope with the loss of container servers without losing data.

You can host more than one container server in a multi-instance integration node. If the active instance of the multi-instance integration node fails, the global cache switches to the container server in the standby instance.

The catalog server controls placement of data and monitors the health of containers. You must have at least one catalog server in your global cache.

## Integration Bus default cache topology



Cache components that are using the default policy are hosted in an integration node with six integration servers

© Copyright IBM Corporation 2016

Figure 13-6. Integration Bus default cache topology

WM676/ZM6761.0

### Notes:

Roles are assigned based on the starting order of the integration servers. For example, the first integration server that starts is a catalog server and a container server.

When an integration node contains one catalog server only, you have a single point of failure. If the catalog server integration server stopped, the cache and data in it are lost.

Restarting the catalog server does not reconnect it to the container servers and data. If you restart the catalog server, you should restart the container integration servers (or the integration node) to give you a new default cache. Data previously in the cache is lost.

The cache requires four ports for each catalog server and three ports for each container.

Common causes of failure are port contention or containers that fail because the catalog failed to start. In all cases, you should resolve the problem (choose a different port range, or restart the catalog server integration server).

## Global cache policy files

- Use a cache policy file to define a grid with multiple integration nodes
- Policy file tells integration node how to participate in global cache
- Specify the policy file as the “policy” property value on all integration nodes that are to participate
- Sample policy files are included in the Integration Bus installation in the server\sample\globalcache directory
  - policy\_multi\_instance.xml
  - policy\_one\_broker\_ha.xml
  - policy\_two\_brokers.xml
  - policy\_two\_brokers\_ha.xml

© Copyright IBM Corporation 2016

Figure 13-7. Global cache policy files

WM676/ZM6761.0

### Notes:

You can configure properties of the embedded global cache by using commands, an XML cache policy file, or the IBM Integration API.

You can customize the default global cache topology to use a specific port range and listener host. You can turn off the default topology and specify your own integration server properties.

Multiple integration nodes can share a cache by using a cache policy file. A policy-file based topology is essentially some default policies that are joined.



## Integration node global cache properties

- Specify policy, if applicable, port range, and listener host
- Integration node generates defaults for port range and listener host name
  - Choose a convenient port range for use by cache components in the integration node
  - Specify listener host name for integration node cache component host name for binding

The screenshot shows the 'TESTNODE\_iibadmin - Integration Node' interface. Under 'Component Properties', the 'Cache Manager' section is highlighted with a red box. It contains the following properties:

Security Cache - Cache Sweep Interval	300
Security Cache - Cache Timeout	60
<b>Cache Manager - Policy</b>	disabled
<b>Cache Manager - Port Range</b>	2840-2859
<b>Cache Manager - Listener Host</b>	

© Copyright IBM Corporation 2016

Figure 13-8. Integration node global cache properties

WM676/ZM6761.0

### Notes:

You can set **Cache Manager** properties at integration node level and integration server level.

At the integration node level, you can set the cache policy, port range, and listener host.

In the IBM Integration web interface, you can set the cache policy (**Policy** property) to none, default, disabled, or the fully qualified name of an XML policy file.

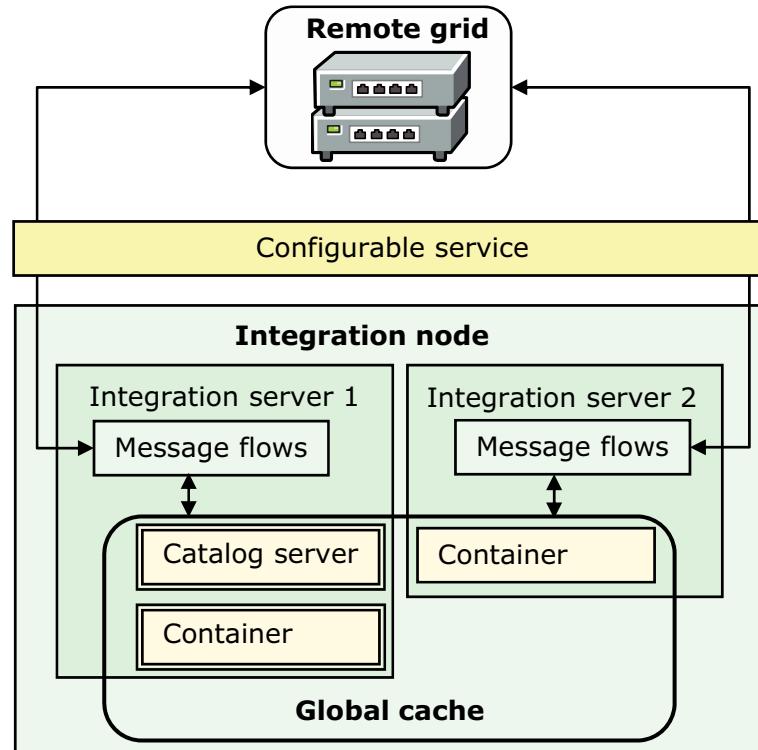
- If you set the **Policy** property to default, the default global cache topology is used.
- If you set the **Policy** property to none, you must set the integration server properties explicitly.
- If you set the **Policy** property to disabled, all cache components in the integration node are not enabled. The cache is not enabled by default.
- If you specify the fully qualified name of a policy file in the **Policy** property, the integration nodes that are listed in the policy file are configured to share the data in the global cache. The path must be absolute, not relative

Specify the port range for the cache components in the **Port Range**. You must specify a minimum of 20 ports in the port range.

Set the **Listener Host** property to the listener host name or IP address that the cache components use.

## Connectivity to WebSphere eXtreme Scale grids

- Can connect to external WebSphere eXtreme Scale grid
- Connections are configured by using the **WXSServer** configurable service
- Connect to multiple external grids, and the embedded global cache at the same time
- Interactions with external grids are logged in Activity Log and resource statistics in the same way as for the embedded global cache



© Copyright IBM Corporation 2016

Figure 13-9. Connectivity to WebSphere eXtreme Scale grids

WM676/ZM6761.0

### Notes:

WebSphere eXtreme Scale provides a scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages data across multiple servers.

As an option, you can use one or more external WebSphere eXtreme Scale grids to store data that you want to reuse. In addition to the grid that is available (as the embedded global cache) within Integration Bus, you can integrate with WebSphere eXtreme Scale grids that are running elsewhere. You can work with multiple external grids, and the embedded grid, at the same time. The figure shows an example configuration.

To connect to an external grid, you need the name of the grid, the host name, and the port of each catalog server for the grid. By using a configurable service to specify the parameters, you can connect to an external WebSphere eXtreme Scale grid. Create a WXSServer configurable service by using the `mgsicreateconfigurableservice` command or in the Integration web interface.

Statistics are available for all activity in the embedded global cache and external WebSphere eXtreme Scale grids. One line of statistics is shown for each configurable service that is used to connect to an external grid.

## Access the global cache by using a JavaCompute node

- Use a JavaCompute node to store or get a key-value pair map in a global cache or an external WebSphere eXtreme Scale grid
- To get a map from the embedded cache, or create the map if it does not exist, add the `MbGlobalMap` object to the JavaCompute node code

Example: `MbGlobalMap globalMap = MbGlobalMap.getGlobalMap("MyMap");`

- When using the embedded cache, use the default map by not specifying a map name

Example: `MbGlobalMap globalMap = MbGlobalMap.getGlobalMap();`

- To get a map from an external grid:
  - Add the `MbGlobalMap` object
  - Specify the name of the map on the external grid and the name of the configurable service that is used to connect to the grid

Example: `MbGlobalMap xc10Map = MbGlobalMap.getGlobalMap("MyMap.LUT", "xc10Connection");`

© Copyright IBM Corporation 2016

Figure 13-10. Access the global cache by using a JavaCompute node

WM676/ZM6761.0

### Notes:

You can use a JavaCompute node to interact with a map in a global cache or an external WebSphere eXtreme Scale grid. You can use a JavaCompute node to store data in a global cache map. You can then create another JavaCompute node to get data from the global cache.

The JavaCompute node uses the `MbGlobalMap` object to access the global cache or external grid. This class can be used to get a key-value pair map, and to put or get data in a global cache map. You cannot create a global cache map explicitly, but if you get a map that does not exist, the map is created automatically. When you get a global map from an external grid, the `getGlobalMap` method makes a connection to the grid if one does not exist. The `MbGlobalMap` object is described in the Java plug-in API documentation.

Interactions with the cache happen outside the message flow transaction, and are committed immediately. If an exception is generated downstream of the node that interacts with the cache, the cache interactions are not rolled back.

For an example of using the JavaCompute node to access a global cache, see “Embedded Global Cache with WebSphere eXtreme Scale” at:

[https://developer.ibm.com/integration/wp-content/uploads/sites/25/2015/02/IIB9000\\_Global\\_Cache\\_WXS.pdf](https://developer.ibm.com/integration/wp-content/uploads/sites/25/2015/02/IIB9000_Global_Cache_WXS.pdf)

## Access data in the cache by using a JavaCompute node

- To put data in the map, you must create a key and value pair in the map
  - Java primitive types and strings are supported for keys and values
  - For the embedded grid only, keys can also be arrays of primitives or strings

Example: `globalMap.put(key, val);`

- To get data that you added to the map, use the `MbGlobalMap` object to get the map and then `globalMap.get()` to get the data

Example:

```
MbGlobalMap globalMap = MbGlobalMap.getGlobalMap("MyMap");  
...  
String val = (String)globalMap.get(key);
```

© Copyright IBM Corporation 2016

Figure 13-11. Access data in the cache by using a JavaCompute node

WM676/ZM6761.0

### Notes:

## Access the global cache by using a Mapping node

- Use a Mapping node to store or get a key-value pair map in an embedded cache or external WebSphere eXtreme Scale grid
- Icons in the Graphical Data Mapping editor toolbar add transforms to the canvas
  - Use a **Cache Put** transform to store data as a key-value pair map in the global cache
  - Use a **Cache Get** transform to retrieve data from the cache for processing or routing
  - Use a **Cache Remove** transform to remove a key-value pair map from the cache
- To handle any exceptions that a Cache transform returns, add a **Cache Failure** transform to the Cache transform group

© Copyright IBM Corporation 2016

Figure 13-12. Access the global cache by using a Mapping node

WM676/ZM6761.0

### Notes:

You can use a Mapping node to interact with a key-value pair map in an embedded cache or external WebSphere eXtreme Scale grid.

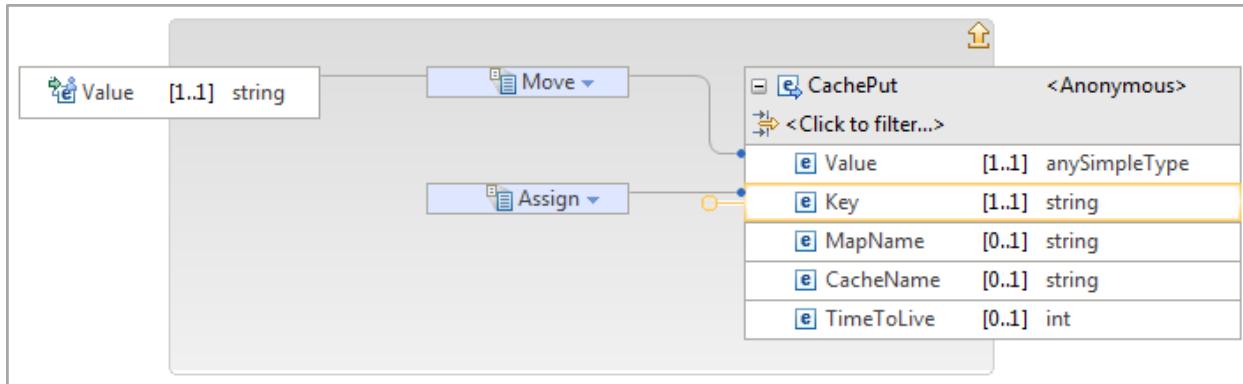


#### Note

This feature requires IBM Integration Bus V10.0.0.2 or later.



## Providing values for key-value pairs in Cache Put



- Provide values for the **Value** and **Key** parameters of the Cache Put transform by using one of the following options
  - Map an element from the input tree to a parameter
  - Set a fixed value for a parameter by right-clicking the parameter in the Cache Put transform, clicking **Add Assign**, and adding the required value in the **Properties** view
  - Set a value for a parameter by using a user-defined property

© Copyright IBM Corporation 2016

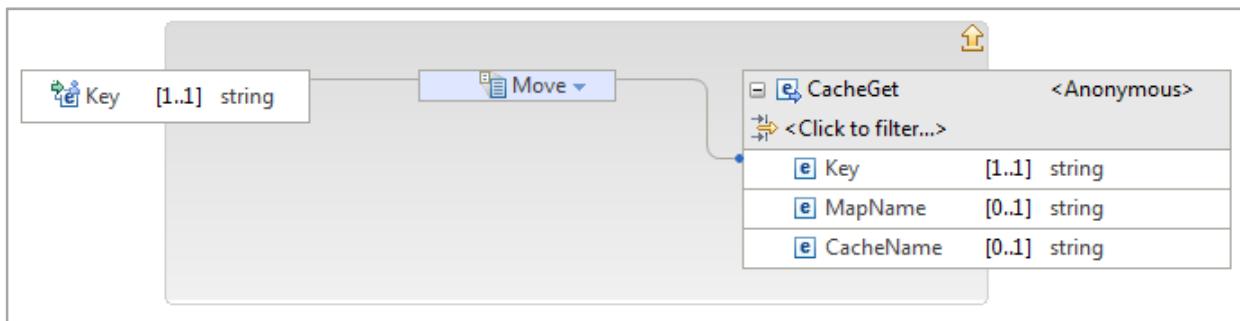
Figure 13-13. Providing values for key-value pairs in Cache Put

WM676/ZM6761.0

### Notes:

In the example, the content for the **Value** parameter is provided by mapping an element that is named **Value**. The content for the **Key** parameter is provided by assigning a fixed value.

## Provide a value for the Cache Get key



- Provide a value for the **Key** parameter of the Cache Get transform by using one of the following options:
  - Map an element from the input tree
  - Set a fixed value for the key by right-clicking the **Key** parameter in the Cache Get transform, clicking **Add Assign**, and adding the required value in the Properties view
  - Set a value for the key by using a user-defined property

© Copyright IBM Corporation 2016

Figure 13-14. Provide a value for the Cache Get key

WM676/ZM6761.0

### Notes:

In this example, the content for the **Key** parameter is provided by mapping an element that is named **Key** from the input tree.

## Global cache administrative tools

Messag...	Timestamp	RM	MSGFLOW	Message Summary	NODE	NODE...
i	BIP11504I	27-Jun-2012 13:25:5...	MF_StoreCache	Waiting for data from input node 'MQ Input'.	MQ Input	INPUT
i	BIP11501I	27-Jun-2012 13:29:3...	MF_StoreCache	Received data from input node 'MQ Input'.	MQ Input	INPUT
i	BIP11109I	27-Jun-2012 13:29:3...	GlobalCache	Connected to cache 'WMB'	Java Compute1	
i	BIP11107I	27-Jun-2012 13:29:3...	GlobalCache	Checked whether key exists in map 'SYSTEM.BROKER.DEFAULTMAP'	Java Compute1	
i	BIP11101I	27-Jun-2012 13:29:3...	GlobalCache	Put data into map 'SYSTEM.BROKER.DEFAULTMAP'	Java Compute1	
i	BIP11107I	27-Jun-2012 13:29:3...	GlobalCache	Checked whether key exists in map 'SYSTEM.BROKER.DEFAULTMAP'	Java Compute1	
i	BIP11101I	27-Jun-2012 13:29:3...	GlobalCache	Put data into map 'SYSTEM.BROKER.DEFAULTMAP'	Java Compute1	
i	BIP11506I	27-Jun-2012 13:29:3...	MF_StoreCache	Committed a local transaction.	MQ Input	INPUT
i	BIP11504I	27-Jun-2012 13:29:4...	MF_StoreCache	Waiting for data from input node 'MQ Input'.	MQ Input	INPUT

- Full resource statistics and activity log in the Integration web interface **Administration Log** provide information about the state of the cache and cache interactions

© Copyright IBM Corporation 2016

Figure 13-15. Global cache administrative tools

WM676/ZM6761.0

### Notes:

You can use the `mqsicacheadmin` command, resource statistics, and the activity log to monitor the global cache. You can use resource statistics and the activity log to monitor external grids.

Activity logs provide a high-level overview of how Integration Bus interacts with external resources. The logs help you to understand what your message flows are doing. Activity log reports when the embedded cache components start.

An integration node collects resource statistics to record performance and operating details of resources that integration servers use. In IBM Integration Explorer, click **Window > Show View > Resource Statistics**. In the **Resource Statistics** view, click the **Global Cache** tab.

The `mqsicacheadmin` command provides information about the global cache that is embedded in an integration node. For example, you can find out the size of a map, list the hosts that are participating in the cache, and clear data from a map.

## Unit summary

Having completed this unit, you should be able to:

- Use the IBM Integration Bus global cache to store data that you want to reuse and share between integration nodes
- Configure properties of the embedded global cache by using commands or an XML cache policy file

© Copyright IBM Corporation 2016

Figure 13-16. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: You cannot define a global cache grid that contains integration nodes on different servers.
  
2. Complete the sentence: In a global cache implementation, a \_\_\_\_\_ server controls the placement of data and monitors the health of containers
  - A. Catalog
  - B. Container
  - C. Integration
  - D. eXtreme Scale grid

© Copyright IBM Corporation 2016

Figure 13-17. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

1.

2.

## Checkpoint answers

1. True or false: You cannot define a global cache grid that contains integration nodes on different servers.

Answer: **False. You can use a cache policy file to define a grid with multiple integration nodes on multiple servers. Place a copy of the same policy file on each computer where an integration node is running.**

2. Complete the sentence: In a global cache implementation, a \_\_\_\_\_ server controls the placement of data and monitors the health of containers

- A. Catalog
- B. Container
- C. Integration
- D. eXtreme Scale grid

Answer: **A**

© Copyright IBM Corporation 2016

Figure 13-18. Checkpoint answers

WM676/ZM6761.0

### Notes:



# Unit 14. Implementing message flow security

## What this unit is about

This unit examines the security considerations in IBM Integration Bus, and how to configure message flow security.

## What you should be able to do

After completing this unit, you should be able to:

- Implement message-level security in a message flow
- Define the differences between administration security, application security, and message transport security
- Reference security profiles in security-enabled message processing nodes
- Use the SecurityPEP node to implement security in a message flow

## How you will check your progress

- Checkpoints
- Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus V10

## Unit objectives

After completing this unit, you should be able to:

- Implement message-level security in a message flow
- Define the differences between administration security, application security, and message transport security
- Reference security profiles in security-enabled message processing nodes
- Use the SecurityPEP node to implement security in a message flow

© Copyright IBM Corporation 2016

---

Figure 14-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- Security overview
- Message security

© Copyright IBM Corporation 2016

---

Figure 14-2. Topics

WM676/ZM6761.0

### Notes:



## 14.1. Security overview

## IBM Integration Bus security overview

- IBM Integration Bus security elements:
  - **Administration security** controls who can access the components of the integration node environment
  - **Message flow (application) security** controls which users can access specific message flows, by reviewing identity credentials that are associated with each message
  - **Message transport security** manages security on the message transports to protect messages that are being sent to and from third parties
- Planning and implementing security is a shared responsibility
- By default, minimal security is enabled

© Copyright IBM Corporation 2016

Figure 14-3. IBM Integration Bus security overview

WM676/ZM6761.0

### Notes:

Securing an IBM Integration Bus environment involves several elements:

- **Administration security** controls who can access the components of the integration node environment itself.
- **Application security** controls which users can access specific message flows, by reviewing identity credentials that are associated with each message.
- **Message transport security** manages security on the message transports themselves to protect messages that are being sent to and from third parties.

This course concentrates on the application security aspect after a brief review of the other security elements.

Planning for a secure IBM Integration Bus environment is a joint effort and shared responsibility between the application designers and system administrators. It is important to understand how the various security elements of IBM Integration Bus, IBM MQ, and external messaging providers must work together to result in a coherent security framework.

By default, only minimal security is enabled during IBM Integration Bus installation. In the exercise environment for this course, you run as a privileged user in a non-secure environment. This unsecured environment is likely to be different from the environment you use in your job.

Securing an IBM Integration Bus environment requires planning. It is better to consider the security needs and plan for them before and during product installation than it is to implement security after message flows are deployed and running.

## IBM Integration Bus administration security

- Controls who can access integration node components
- Established for each integration node
- Administration security includes these components:
  - Integration servers
  - Connections between Integration Toolkit and the integration node environment
  - Connections between the Integration web interface and the integration node environment
  - Users of the Integration API
  - A subset of the `mqsi` commands
- Integration Bus administrator is typically responsible for administration security, but planning involves developers

© Copyright IBM Corporation 2016

Figure 14-4. IBM Integration Bus administration security

WM676/ZM6761.0

### Notes:

IBM Integration Bus administration security regulates who can access the components of an integration node and its environment. The Integration Bus administrator enables administration security, and then grants privileges to individuals or groups of users to give them the appropriate level of authority to complete tasks.

The components of an integration node environment include:

- The integration servers that the integration node uses.
- The connections between the user of the Integration Toolkit and the integration node environment.
- The connections between the user of the IBM Integration web interface and the integration node environment.
- The users of the IBM Integration API. The IBM Integration API is the common API that Integration Bus uses to pass commands to the integration node environment.

- A subset of the `mqsi` commands, specifically those commands that allow the user to review or alter the integration node configuration. These commands are:

```
mqsiclangeresourcestats  
mqsicreateexecutiongroup  
mqsideleteexecutiongroup  
mqsideploy  
mqsilist  
mqsimode  
mqswireloadsecurity  
mqswireportresourcestats  
mqsistartmsgflow  
mqsistopmsgflow  
mqsiwebuseradmin
```

The Integration Bus administrator generally manages administration security. However, since administration security can affect how a developer must interact with the integration node environment, planning for administration security is often a shared responsibility during the planning phases.

IBM course WM646, *IBM Integration Bus V10 System Administration* explains the details of planning and implementing administration security.

## Message transport security

- Protects messages that are delivered to and from message flows on various message transports
- Includes restricting access to IBM MQ and JMS queues
- Configuration is typically a shared responsibility between IBM Integration Bus administrator, network administrator, and in some implementations the IBM MQ administrator
- Application developers must understand the ramifications of transport security because it can affect application design and implementation
  - WS-Security
  - HTTP and HTTPS transport
  - Encrypted messages
  - Digital certificates and signatures

© Copyright IBM Corporation 2016

Figure 14-5. Message transport security

WM676/ZM6761.0

### Notes:

The messages that message flows use or produce can have their own protection requirements. It is normally the responsibility of the transport provider to secure those messages while in transit. This protection requirement can affect both the Integration Bus administrator and Integration Bus developers.

For example, it might be necessary for the administrator to configure the Integration Bus environment to match the message transport security requirements. Securing IBM MQ queues and JMS queues and messaging points can be part of this requirement.

It might be necessary for application developers to configure message flows to handle transport security features. For example, it might be necessary to configure HTTP nodes for HTTPS or SOAP nodes for WS-Security, how to manage encrypted messages, digital signatures, and other security elements that are related to message transport.

It is important for administrators and application developers to understand and plan jointly for the security needs of message transports.

## Message flow security overview

- Controls access to individual messages based on the user credentials that are part of the message; different transports provide different types of credentials
  - Most message transports provide a user ID (“identity token”)
  - Some message transports provide a password (“security token”)
  - Other tokens can be included with the message
- Primary components of message flow security
  - **Authentication:** User ID in message that is known to Integration Bus
  - **Authorization:** User ID that is allowed to use this message flow
  - **Credential mapping:** Translating the tokens to an alternative form or alias
- Application security involves multiple components
  - Security-enabled message processing nodes
  - Message flow security manager
  - Security profiles
  - External security providers

© Copyright IBM Corporation 2016

Figure 14-6. Message flow security overview

WM676/ZM6761.0

### Notes:

The remainder of this unit describes message flow application security. Integration Bus message flow application security primarily concerns the credentials that are associated with a message that is submitted for processing in a message flow. These credentials can include a user ID, password, or both. The credential information can be included within the message body, or as part of the message headers.

Message flow security determines whether the submitter of a message (that the security token identifies) is allowed to submit the message for processing. Determining this permission can involve a combination of steps:

- **Authentication** establishes the identity of a user or entity and verifies that the identity is valid. For example, if the user ID for a message is USER123, authentication determines that USER123 is known to Integration Bus by verifying the identity token for the message. You use an *external security provider* to verify the token.
- **Authorization** determines that the user or entity that is associated with the identity token is allowed to use the message flow.

- **Credential mapping** is the process of mapping (translating) the credentials in the incoming security token to a different format or to an alternative identity. Identity mapping, sometimes called *identity federation*, can also map identity information in one realm to an equivalent identity in a different realm. For example, a user can be identified with user ID USER123 in the Sales application, but FCST821 in the forecasting system. Credential mapping can translate the various identities that are associated with the same user.

These three component checks are normally done in sequence:

1. The user ID is authenticated (if requested).
2. The security credentials are mapped (if requested).
3. Either the original (source) identity information or mapped identity information is checked to see whether it is authorized (if requested).

Many components are important for application security:

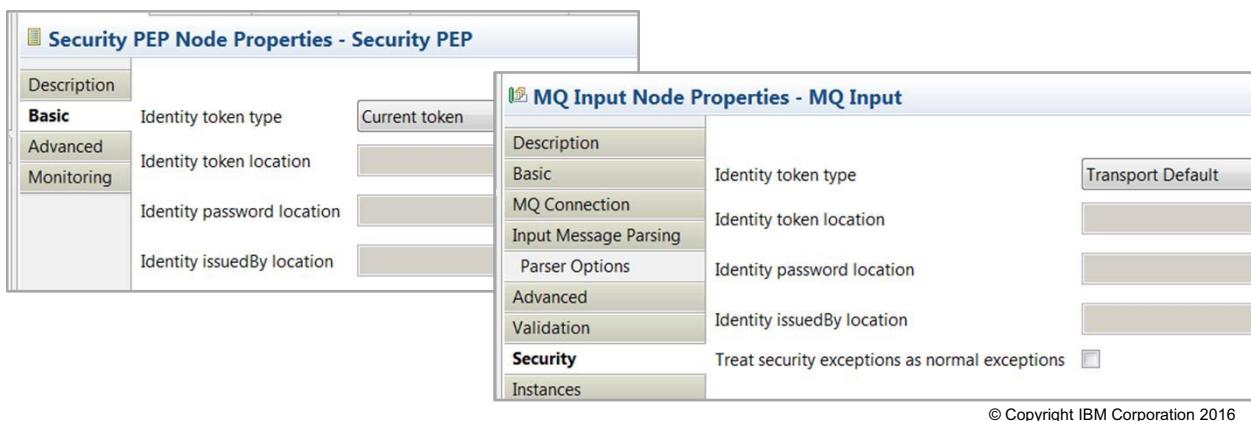
- **Security-enabled message processing nodes** are message flow nodes that can query the security information in a message and then act based on that information.
- The **message flow security manager** works with the integration node to act on security information, as the security profiles dictate.
- **Security profiles** control what actions to take (for example, authenticate, authorize, or map credentials) when a security-enabled message processing node receives a message.
- When the security profiles so direct, the message flow security manager calls **external** security providers to authenticate, authorize, or map credentials.

Several message processing nodes can be used in managing message flow security. These nodes are used in combination with the message flow security manager and security profiles to authenticate, authorize, and map credentials.

## 14.2.Message security

## Security-enabled message processing nodes

- To start message flow security processing at run time, at least one node in the message flow must be *security-enabled*
  - Security-enabled input nodes: MQ Input, HTTP Input, SOAP Input, SCA Input, SCA AsyncResponse
  - Security-enabled output and request nodes that support identity propagation: MQ Output, HTTP Request, SOAP Request, SOAP AsyncRequest, SCA Request, SCA AsyncRequest
  - Security Policy Endpoint (Security PEP) node



© Copyright IBM Corporation 2016

Figure 14-7. Security-enabled message processing nodes

WM676/ZM6761.0

### Notes:

To use message flow security, at least one message processing node in a message flow, must be *security-enabled*. When a message encounters a security-enabled message processing node, security processing begins.

The subset of message processing nodes on which you can enable security are:

- Input nodes: MQ Input, HTTP Input, SOAP Input, SCA Input
- Output and request nodes: MQ Output, HTTP Request, SOAP Request, SOAP AsyncRequest
- SecurityPEP (Security Policy Endpoint)

In general, the *input* node that starts the message flow is where initial security checking is done. Initial security checking includes authentication, authorization, and credential mapping. The *output* nodes are used to propagate the security credentials through the message transport to the message destination.

You can include one or more SecurityPEP nodes anywhere in the message flow to use an external security provider to authenticate, authorize, or map credentials.

You enable security by configuring the node properties (in most cases, on the **Security** tab). You set the properties to control the extraction of the identity information from the message (where a security profile is also associated with the node).

For example, the MQ Input node **Security** properties include the following properties:

- **Identity token type** specifies what type of identity token is present in the message. Valid values are:
  - User name.
  - User name + password.
  - SAML assertion.
  - X.509 certificate.
  - Transport default. When this value is set, the node retrieves the **UserIdentifier** element from the IBM MQ message descriptor (MQMD) and puts it into the **Properties.IdentitySourceToken** element. At the same time, the node sets the **Identity Source Type** element to `username` and sets the **Identity Source Issued By** element to the PUT application name.
- **Identity token location** specifies where in the message to find the identity (user ID). You can use an XPath, ESQL, or literal expression to identify the location.
- **Identity password location** specifies where in the message to find the password. You can use an XPath, ESQL, or literal expression to identify the location. This property can be set only if the **Identity token type** is set to `Username + Password`.
- **Identity issuedBy location** specifies who sent the identity. This value specifies the issuer that can be passed to a WS-Trust V1.3 STS provider.
- **Treat security exceptions as normal exceptions**: If this property is enabled, any security exception causes the message assembly to propagate through the **Failure** terminal of the node (if it is wired). The default value is disabled, which ensures that a security exception forces a rollback, even if the **Failure** terminal is wired.

The security properties vary with each node. For example, some security properties on the HTTP Input node do not exist on the MQ Input node because of the differences in the transport security.

Be sure that you understand the security properties that are specific to the type of message processing node that you are using. For example, when you use an MQ Input node, you can retrieve a user ID from the MQMD, but you have no provision for storing a password in the MQMD. A user ID and password are both required to authenticate the credentials. Without any authentication, you must trust the MQ Input message that is based only on the user ID. Otherwise, you must extract the password from within the message itself (not from the MQMD) to authenticate.

## Message flow security manager

- Security manager that is supplied with Integration Bus
- Not enabled by default when Integration Bus is installed
  - Security relegated to message transports
  - All messages that reach message flow are processed
- When security-enabled message processing node is encountered, message flow security manager consults security policy to determine whether security processing is required
- Allows integration node to:
  - Extract the identity credentials from an inbound message
  - Authenticate the identity by calling external security provider
  - Map the identity to alternative identity by calling external security provider
  - Authenticate original or alternative identity by calling external security provider
  - Propagate original or alternative identity (or both) in the outbound message

© Copyright IBM Corporation 2016

Figure 14-8. Message flow security manager

WM676/ZM6761.0

### Notes:

If the Integration Bus administrator does not enable message flow security, Integration Bus relies on the security that the message transport mechanism provides. In this case, the integration node processes **all** messages that are delivered to it, using the integration node service identity (the user ID that the integration node runs under) as a proxy identity for all message instances. Any identity that is present in the incoming message is ignored.

When the message flow security manager is enabled, it does not delegate security to the message transport mechanism. Instead, the message flow security manager enables the integration node to:

- Extract the identity from an inbound message
- Authenticate the identity (by using an external security provider)
- Map the identity to an alternative identity (by using an external security provider)
- Check that either the alternative identity or the original identity is authorized to access the message flow (by using an external security provider)
- Propagate the original identity, alternative identity, or both in the outbound message

By default, the message flow security manager is not enabled when Integration Bus is installed. The Integration Bus administrator must enable the message flow security manager. The administrator generally enables the message flow security manager when the security profiles are created and deployed.

## Security profiles

- Define security operations to complete when security-enabled message node is encountered (authenticate, authorize, map credentials, propagate identity)
  - Identity propagation passes the source or mapped security credentials in the outbound message through specific output or request nodes
  - Not all nodes support identity propagation
- Specify external security provider to call
- Integration Bus administrator creates security profiles
  - IBM Integration web interface
  - `mqsicreateconfigurableservice` command
- Integration Bus developer can set security profiles on message flow and message processing nodes
- Requires that you stop and start the integration server for a security profile property change to take effect

© Copyright IBM Corporation 2016

Figure 14-9. Security profiles

WM676/ZM6761.0

### Notes:

As part of configuring security, the Integration Bus administrator must create and deploy a security profile. A security profile defines the security operations that a message flow must do when the message flow contains a security-enabled input or output node or a SecurityPEP node. The message flow security manager accesses the security policy at run time.

The security profile also specifies whether **identity propagation** is required. Identity and security token propagation enables the identity and security tokens that are associated with each message to be propagated throughout a message flow, and onto target applications through output or request nodes. Identity propagation provides a means for the tokens, which were authenticated, authorized, or mapped (if the security profile requested those actions) to be passed on with the outbound messages.

The outbound nodes that support identity propagation are: CICS Request, HTTP Request, IMS Request, MQ Output, SAP Request, SCA AsyncRequest, SCA Request, SOAP AsyncRequest, and SOAP Request.

A security profile allows an integration node administrator to specify whether security processing is done on the identity or security tokens that are associated with messages in the message flow. The

security profile also specifies the external security provider (also known as a *policy decision point*) that the integration node uses.

The Integration Bus administrator uses the IBM Integration web interface or the `mqsicreateconfigurableservice` command to create a security profile.

The Integration Bus developer can specify the security profile on a message flow and a node in the BAR file editor in the Integration Toolkit.

## SecurityProfiles configurable service

- **mapping** defines the type of source identity mapping
- **propagation** enables identity propagation on output and request nodes
- **passwordValue** defines how passwords are treated when they enter a message flow
- **authorizationConfig** defines how the integration node connects to the provider and checks access
- **authenticationConfig** defines the information that the integration node needs to connect to the provider and look up the identity tokens
- **idToPropagateToTransport** enables the use of a specific security identity for propagation

Properties	
mapping	NONE
rejectBlankpassword	FALSE
propagation	TRUE
passwordValue	PLAIN
keyStore	Reserved for future use
authorizationConfig	
authenticationConfig	
idToPropagateToTransport	Message ID
trustStore	Reserved for future use
authentication	NONE
authorization	NONE
mappingConfig	
transportPropagationConfig	

© Copyright IBM Corporation 2016

Figure 14-10. SecurityProfiles configurable service

WM676/ZM6761.0

### Notes:

The Integration Bus administrator uses the IBM Integration web interface or the `mqsicreateconfigurableservice` command to create a security profile.

A security profile contains values for the following properties:

- The **authentication** property defines the type of authentication on the source identity.
- The **authenticationConfig** property defines the provider-specific configuration string that the integration node needs to connect to the provider and look up the identity tokens.
- The **mapping** property defines the type of mapping on the source identity.
- The **mappingConfig** property defines a provider-specific configuration string that the integration node needs to connect to the provider and look up the mapping routine.
- The **authorization** property defines the types of authorization checks that are done on the mapped or source identity.
- The **authorizationConfig** property defines how the integration node connects to the provider, and contains additional information that can be used to check access (for example, a group that can be checked for membership). It is a provider-specific configuration string.

- The **passwordValue** property defines how passwords are treated when they enter a message flow.
- The **propagation** property enables identity propagation on output and request nodes.
- The **idToPropagateToTransport** property enables the use of a specific security identity for propagation.
- The **transportPropagationConfig** property provides a specific security identity to propagate when **idToPropagateToTransport** is set to STATIC\_ID.

For a detailed description of each property, see the IBM Knowledge Center for IBM Integration Bus V10.

## Display security profile properties

- Using the IBM Integration web interface or the `mqsi reportproperties` command with the `-c SecurityProfiles` parameter

Example command to display all security profiles on an integration node

```
mqsi reportproperties IBNODE -c SecurityProfiles
-o AllReportableEntityNames -r
ReportableEntityName=''
SecurityProfiles
  Default_Propagation=''
    authentication = 'NONE'
    authenticationConfig = ''
    authorization = 'NONE'
    authorizationConfig = ''
    idToPropagateToTransport = 'Message ID'
    keyStore = 'Reserved for future use'
    mapping = 'NONE'
    mappingConfig = ''
    passwordValue = 'PLAIN'
    propagation = 'TRUE'
    rejectBlankpassword = 'FALSE'
    transportPropagationConfig = ''
    trustStore = 'Reserved for future use'
```

© Copyright IBM Corporation 2016

Figure 14-11. Display security profile properties

WM676/ZM6761.0

### Notes:

You can use the IBM Integration web interface and the `mqsi reportproperties` command to view security profile properties.

## Setting security profiles in the BAR file

The screenshot shows the 'Manage' view of a BAR file. The left pane lists resources: BusinessRulesApp (Application), application.descriptor (Nov 23, 2015 1:55:36 PM), BookOrderPostage.msgflow (Message flow), and BookOrderPostage (Decision Service). Under BookOrderPostage, MQ Input and MQ Output are listed. The MQ Input node is selected and highlighted with a red box. The right pane shows the 'MQ Input' configuration panel with tabs for 'Configure', 'Workload Management', and 'z/OS serialization token'. The 'Configure' tab is active, and the 'Security profile' field is highlighted with a red box.

- Can specify security profile at the message flow level and at the message processing node level
  - If **Security profile** is not specified on the node, the node uses the security profile that is set on the message flow
  - If **Security profile** is blank on both the node and the message flow, security is not enabled
  - If **Security profile** is set to **No Security** on a node, security is not enabled for that node
  - If **Security profile** is set to a security profile name, that profile determines the message flow security

© Copyright IBM Corporation 2016

Figure 14-12. Setting security profiles in the BAR file

WM676/ZM6761.0

### Notes:

The message flow, the security-enabled input and output message processing nodes, and the SecurityPEP node have a **Security profile** property that you can set with the BAR File editor. The **Security Profile** property can be set as follows:

- The **Security Profile** property can be **blank**, which causes the node to inherit the **Security Profile** property that is set at the message flow level. If the **Security Profile** property is blank at both the node level and the message flow level, security is not enabled for the node.
- The **Security Profile** property can be set to **No Security**, which explicitly disables security for that node.
- The **Security Profile** property can be set to the name of a security profile. When a security profile name is set, that profile determines the message flow security. If the named security profile does not exist in the runtime environment, the message flow fails to deploy. If the specified external security provider does not support the type of token that is configured on the node for the security operation, an error is reported and the message flow fails to deploy.

## External security providers

- Third-party applications that can authenticate, authorize, and map credentials
- Integration Bus supports:
  - WS-Trust v1.3 compliant Security Token Service with IBM Tivoli Federated Identity Manager V6.2 for authentication, authorization, and mapping
  - IBM Tivoli Federated Identity Manager V6.1 for authentication, authorization, and mapping (for compatibility with previous versions of Integration Bus and WebSphere Message Broker)
  - Lightweight Directory Access Protocol (LDAP) for authentication and authorization
  - Windows Domain Controllers for authentication
  - Kerberos Key Distribution Center (KDC) for authentication
- Security profile can specify multiple external security providers
- Integration Bus administrator configures external security providers with application developers

© Copyright IBM Corporation 2016

Figure 14-13. External security providers

WM676/ZM6761.0

### Notes:

If an identity token or security token must be authenticated, authorized, or credential mapped (as the security policy directs), IBM Integration Bus calls an **external service** provider. The default security profile (named *Default Propagation*) does not reference an external security provider, and can propagate only the security information through the message flow.

IBM Integration Bus supports these external security providers:

- WS-Trust V1.3 compliant Security Token Service (STS) with IBM Tivoli Federated Identity Manager V6.2 is supported for authentication, authorization, and mapping.
- IBM Tivoli Federated Identity Manager V6.1 is supported for authentication, authorization, and mapping.
- Lightweight Directory Access Protocol (LDAP) is supported for authentication and authorization only.
- Windows Domain Controllers is supported for authentication.
- Kerberos KDC is supported for authentication.

**Note**

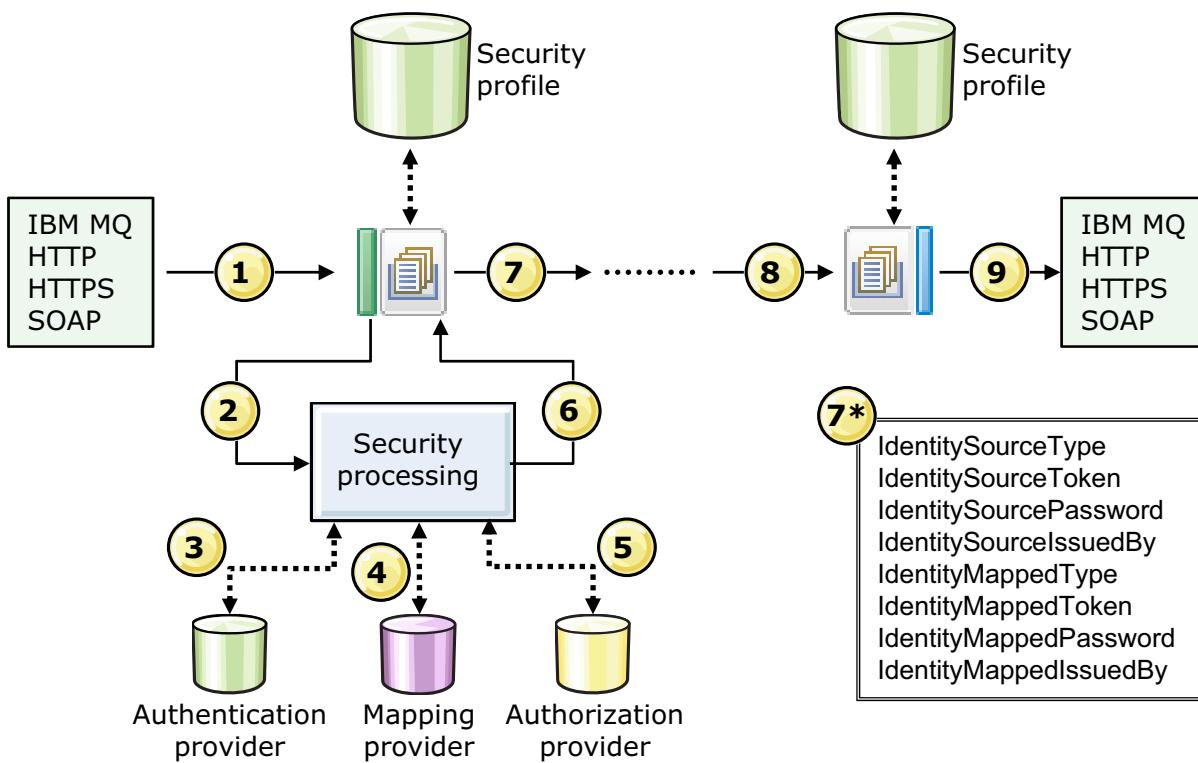
Support for IBM Tivoli Federated Identity Manager V6.1 is included for compatibility with previous versions of IBM Integration Bus. If possible, upgrade to IBM Tivoli Federated Identity Manager V6.2.

The IBM Integration Bus administrator can configure the security profile to use different security providers for different security functions. For example, you might use LDAP for authentication and WS-Trust V1.3 STS for mapping and authorization.

If you do not use an external service provider but you must authenticate, authorize, or map credentials, you must write the logic for those operations yourself. Without an external security provider, IBM Integration Bus by itself can propagate only identity information.

The IBM Integration Bus administrator is responsible for configuring the external security providers, with application developer requirements. Because this configuration process is an administrator responsibility, and because of the complexity of configuring these providers, configuration details are omitted in this course. For more information about configuring the providers, see the IBM Knowledge Center for IBM Integration Bus V10.

## Security processing details: Flow diagram



© Copyright IBM Corporation 2016

Figure 14-14. Security processing details: Flow diagram

WM676/ZM6761.0

### Notes:

The last several pages explained the fundamentals of message security processing and the components that participate in the process. The next pages show the details of security processing that occur at run time.

The example describes security processing for a message flow that begins with an MQ Input node and ends with an MQ Output node. The numbers on the diagram correspond to the steps on the next four figures.

This diagram changes if the message flow uses different transport protocols or includes other security-enabled message processing nodes, but the basic processing stays the same.

You examine the use of the SecurityPEP node later in this unit.

## Security processing details (1 of 4)

1. Message arrives at security-enabled input node
  - If a security profile that is associated with node is found, call message flow security manager to read the profile
  - Determine security actions and the external security provider
  - For SOAP node, you might not need to call message flow security manager; you can implement some WS-Security functions without it
  
2. Security-enabled input node extracts identity information from message
  - Node **Security** properties dictate where in message or message header to find identity information
  - Sets **Properties.IdentitySource** elements
  - If identity information cannot be retrieved, raise a security exception

© Copyright IBM Corporation 2016

Figure 14-15. Security processing details (1 of 4)

WM676/ZM6761.0

### Notes:

1. When a message arrives at a security-enabled input node (MQ, HTTP, SCA, or SOAP), the presence of a security profile that is associated with the node indicates whether message flow security is configured. The integration node's security manager is called to read the profile, which specifies the combination of propagation, authentication, authorization, and mapping for the identity of the message. It also specifies the external security provider.
  
2. If a security profile is associated with the node or message flow, the security-enabled input node extracts the identity information from the input message (based on the configuration on the node's **Security** properties page). It sets the **IdentitySource** elements in the **IdentitySource** folder. If the security tokens cannot be successfully extracted, a security exception is raised.

The **IdentitySource** elements in the **IdentitySource** folder include:

- IdentitySourceType
- IdentitySourceToken
- IdentitySourcePassword
- IdentitySourceIssuedBy

## Security processing details (2 of 4)

3. If security profile requests authentication, message flow security manager calls configured external security provider
  - Authentication result is returned to security cache
  - If authentication cannot be completed, then return security exception to input node
4. If security profile requests identity mapping, message flow security manager calls configured external security provider
  - Mapped identity information is returned to security cache
  - Sets **Properties.MappedSource** elements
  - If authentication cannot be completed, then return security exception to input node
5. If security profile requests authorization, message flow security manager calls configured external security provider
  - Authentication result is returned to security cache
  - If authentication cannot be completed, then return security exception to input node

© Copyright IBM Corporation 2016

Figure 14-16. Security processing details (2 of 4)

WM676/ZM6761.0

### Notes:

3. If authentication is specified in the security profile, the security manager calls the configured security provider to authenticate the identity. A failure returns a security exception to the node. A security cache is provided for the authentication result, which enables subsequent messages (with the same credentials) arriving at the message flow to be completed with the cached result, unless it expired.
4. If identity mapping is specified in the security profile, the security manager calls the configured security provider to map the identity to an alternative identity. A failure returns an exception to the node. Otherwise, the mapped identity information is set in the **IdentityMapped** elements in the **Properties** folder. A security cache is provided for the result of the identity mapping.
5. If authorization is specified in the security profile, the security manager calls the configured security provider to authorize that the identity (either mapped or source) has access to this message flow. A failure returns an exception to the node. A security cache is provided for the authorization result.

## Security processing details (3 of 4)

6. Control returns to the input node
  - If security exception was raised, then back out message and end message flow transaction
  - Do not log error or send the message assembly through the **Failure** terminal (unless **Treat security exceptions as normal** node property is set)
7. The message, which includes the **Properties** folder and source and mapped identity elements, is propagated through the **Out** terminal

© Copyright IBM Corporation 2016

Figure 14-17. Security processing details (3 of 4)

WM676/ZM6761.0

### Notes:

6. When all security processing is complete, or when the message flow security manager raises an exception, control returns to the input node.
7. If all security processing is successful, the message, including the **Properties** folder and its source and mapped identity information, is propagated down the message flow through the **Out** terminal of the input node.



## Security processing details (4 of 4)

8. If the message reaches a security-enabled output or request node, and a security profile for the node directs the node to propagate the identity information:
  - Use *mapped* identity from the **Properties** folder
  - If mapped identity is not set, or has a token type that the output node does not support, then use *source* identity from **Properties** folder
  - If neither identity is set, or neither identity has a token type that the output node supports, then raise a security exception
9. Send propagated identity information (in the message header) from the output or request node

© Copyright IBM Corporation 2016

Figure 14-18. Security processing details (4 of 4)

WM676/ZM6761.0

### Notes:

8. When the message reaches a security-enabled output or request node, a security profile (with propagation enabled) associated with the node indicates that the current identity token is propagated when the message is sent.

If the security profile indicates that propagation is required, the mapped identity is used. If the mapped identity is not set, or if it has a token type that the node does not support, the source identity is used. If no identity is set, or if the mapped and the source identity has a token type that the node does not support, a security exception is returned to the node.

Some output and request nodes can automatically propagate only the identity token. If you use one of those nodes and you want to include the security token in the outbound message, use a Compute node in the message flow before the output or request node. Save the security token in the message header or message body, depending on what the receiving application expects.

9. The propagated identity is included in the appropriate message header when it is sent.

## More about security processing

- If the message flow includes other security-enabled nodes, then security processing works similarly, depending on security policy and node properties
- Security cache can reduce number of calls to external security providers
  - Cache contains the results of all calls to external security providers
  - If the cached value can be used, no call is made to an external security provider
  - Integration Bus administrator can use the `mqssireloadsecurity` command to expire some or all cached values
  - Integration Bus administrator can use the `mqsicchangeproperties` command to configure automatic expiration of cached values
- Configuring properties of security-enabled input nodes depends on the message transports that the nodes use
- Configuring security for z/OS integration nodes is different than for integration nodes that run on distributed operating systems

© Copyright IBM Corporation 2016

Figure 14-19. More about security processing

WM676/ZM6761.0

### Notes:

If you add other security-enabled nodes or SecurityPEP nodes to the message flow, the basic processing remains the same.

Each message that a message flow processes can make many calls to external security providers. To reduce the number of calls, a security cache holds the results of all calls to external security providers. Instead of immediately calling the external security provider, the integration node checks to see whether the lookup values are already cached for the security identity and security token. If the values are in cache and are still valid, those cached values are used.

You can use the `mqssireloadsecurity` line command to force some or all of the cached values to be marked as “expired”. You can also use the `mqsicchangeproperties` line to configure the automatic expiration.

How you configure security-enabled input nodes depends on the message transport that is being used. For example, SOAP nodes do not have **Security** properties that you configure. Security for SOAP nodes is configured by using WS-Security (or Kerberos) policy sets and bindings.

Configuring security is different for integration nodes that are running in a z/OS environment as compared with integration nodes that are running in other environments.

## Using the SecurityPEP node



- Does the same authentication, authorization, and credential mapping functions as a security-enabled input node
- Use one or more in a message flow when:
  - Message flow requires security processing but message flow does not start with security-enabled input node
  - Message flow starts with security-enabled input node, but you do not want to do all security processing in that node (for example, to authenticate on the input node only)
  - You must map credentials more than one time in a message flow (for example, to propagate different mapped credentials to different output or request nodes)
- Security processing works the same in SecurityPEP node as security-enabled input node
  - Be aware of where the identity credentials exist in the incoming message
- Security exceptions are propagated to **Failure** terminal

© Copyright IBM Corporation 2016

Figure 14-20. Using the SecurityPEP node

WM676/ZM6761.0

### Notes:

When you use the Security Policy Endpoint (SecurityPEP) message processing node in a message flow, it can do the same authentication, authorization, and credential mapping that a security-enabled input node does. It calls the message flow security manager just as an input node does.

You might want to use one or more SecurityPEP nodes in a message flow for several reasons:

- If the message flow does not start with a security-enabled input node; for example, if the message flow starts with a JMS Input or File Input node.
- If the message flow starts with a security-enabled input node, but you want only to authenticate or authorize the credentials on the input node, and not map credentials at that point.
- If the message flow starts with a security-enabled input node, but you want only to authenticate or authorize the credentials on the input node. Later in the flow you must map credentials multiple times. This scenario might be the case if the message flow contains multiple security-enabled output or request nodes and you must map credentials before each of those nodes. This capability is useful if you are sending a message to multiple destinations by using

multiple message transports, for example. Use a SecurityPEP node that is configured before each output or request node.

The authentication, authorization, and credential mapping works in the SecurityPEP node as it does in a security-enabled input node. However, you must understand where the identity token (and security token, if applicable) is in the inbound message. These locations can affect how you configure the SecurityPEP node. For example, if the message flow does not contain a security-enabled input node, the **IdentitySource** and **IdentityMapped** elements of the **Properties** tree of the inbound message are empty.

A security exception in the SecurityPEP node propagates the message assembly to the **Failure** terminal of the node, and writes a message to the system log.

The SecurityPEP node is in the **Security** drawer of the Message Flow editor.

## Security exception handling

- When a security processing exception occurs on a security-enabled input node, the node:
  1. Rolls back the message to the message transport
  2. Ends the message flow
- By default, no message is written to the system log
  - The message assembly is not propagated through the **Failure** terminal to prevent a security denial of service attack from filling the logs and destabilizing the system
- If the input node **Treat security exceptions as normal** property is set, then a security exception is handled like any other exception

© Copyright IBM Corporation 2016

Figure 14-21. Security exception handling

WM676/ZM6761.0

### Notes:

Exception handling for security exceptions is not managed the same as most other input node errors. Most exceptions that occur on input nodes cause the node to:

1. Roll back the message to the message transport
2. Write an error message to the system log
3. Propagate the message assembly to the input node **Failure** terminal

When a security processing exception occurs on a security-enabled input node, the node:

1. Rolls back the message to the message transport
2. Ends the message flow

By default, no message is written to the system log, and the message assembly is not propagated through the Failure terminal. Security exceptions in security-enabled input nodes are managed in this way to prevent a security denial of service attack from filling the logs and destabilizing the system.

If the input node **Treat security exceptions as normal** property is set, then a security exception is handled like any other exception.

## Diagnosing security problems

- Option 1: Select the **Treat Security Exceptions as normal exceptions** property on the input nodes
- Option 2: Run a user trace to find out why access to a secured message flow was denied
  1. To ensure that the reason codes that are returned from the security provider are displayed in the traced exception, use the `mqsireloadsecurity` command to clear the security cache.
  2. Enable user trace for the message flow.
  3. Resend the request that the security provider rejected.
  4. Stop the user trace.
  5. Use the `mqsireadlog` command to examine the integration node and security provider trace information.

© Copyright IBM Corporation 2016

Figure 14-22. Diagnosing security problems

WM676/ZM6761.0

### Notes:

By default, security exceptions that occur in an input node are not processed in the same way as other errors (see “Security exception processing”: [http://www.ibm.com/support/knowledgecenter/SSKM8N\\_8.0.0/com.ibm.etools.mft.doc/ap04080\\_.htm](http://www.ibm.com/support/knowledgecenter/SSKM8N_8.0.0/com.ibm.etools.mft.doc/ap04080_.htm)). Security exceptions are not logged to the system event log to prevent a security denial of service attack from filling the logs and destabilizing the system.

So, by default, you cannot diagnose input node security exceptions in the same way as other errors.

## Unit summary

Having completed this unit, you should be able to:

- Implement message-level security in a message flow
- Define the differences between administration security, application security, and message transport security
- Reference security profiles in security-enabled message processing nodes
- Use the SecurityPEP node to implement security in a message flow

© Copyright IBM Corporation 2016

---

Figure 14-23. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions (1 of 2)

1. True or false: By default, IBM Integration Bus processes all messages (that is, no security is enabled by default).
2. Which of these tasks can a security-enabled input node do:
  - a. Use an ODBC or JDBC connection to do a database lookup
  - b. Use an external security provider to authenticate credentials
  - c. Reload the entire security cache
  - d. Use an external security provider to authorize credentials
  - e. Use an ESQL or XPath statement to map credentials without using an external security provider
3. True or false: If an error is detected when an external security provider is being called, the message flow always rolls back.

© Copyright IBM Corporation 2016

Figure 14-24. Checkpoint questions (1 of 2)

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint questions (2 of 2)

4. Which of the following components are needed to authenticate security credentials at run time?
  - a. A message flow that contains a security-enabled input node
  - b. The deployed and enabled message flow security manager
  - c. A configured external security provider
  - d. A configured and deployed security profile
  - e. At least one SecurityPEP node in the message flow

© Copyright IBM Corporation 2016

Figure 14-25. Checkpoint questions (2 of 2)

WM676/ZM6761.0

### Notes:

Write your answer here:

4.

## Checkpoint answers (1 of 2)

1. True or false: By default, IBM Integration Bus processes all messages (that is, no security is enabled by default).

Answer: **True**

2. Which of these items can a security-enabled input node do:

- a. Use an ODBC or JDBC connection to do a database lookup
- b. Use an external security provider to authenticate credentials
- c. Reloading the entire security cache
- d. Use an external security provider to authorize credentials
- e. Use an ESQL or XPath statement to map credentials without using an external security provider

Answer: **b** and **d**

3. True or false: If an error is detected when an external security provider is being called, the message flow always rolls back.

Answer: **False. The error handling behavior depends on the message flow and the configuration.**

© Copyright IBM Corporation 2016

Figure 14-26. Checkpoint answers (1 of 2)

WM676/ZM6761.0

### Notes:



## Checkpoint answers (2 of 2)

4. Which of the following components are needed to authenticate security credentials at run time?
    - a. A message flow that contains a security-enabled input node
    - b. The deployed and enabled message flow security manager
    - c. A configured external security provider
    - d. A configured and deployed security profile
    - e. At least one SecurityPEP node in the message flow
- Answer: **b, c, and d**

© Copyright IBM Corporation 2016

Figure 14-27. Checkpoint answers (2 of 2)

WM676/ZM6761.0

### Notes:

## Exercise 7



### Implementing IBM Integration Bus runtime security

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 14-28. Exercise 7

WM676/ZM6761.0

#### Notes:

In this exercise, you implement the IBM Integration Bus runtime security to allow security credentials to be propagated in a message flow.



## Exercise objectives

After completing this exercise, you should be able to:

- Propagate security credentials within a message flow
- Configure message flow nodes to enable secure message processing
- List the types of security tokens
- Use a Compute node to simulate the actions of an external security provider

© Copyright IBM Corporation 2016

---

Figure 14-29. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.

# Unit 15. Implementing publish/subscribe

## What this unit is about

IBM Integration Bus uses publish/subscribe to notify applications of significant events that occur in integration nodes. IBM Integration Bus can also act as a content filtering provider for IBM MQ to allow subscribers to refer to elements in the body of publications. This unit describes how IBM Integration Bus supports these publish/subscribe applications.

## What you should be able to do

After completing this unit, you should be able to:

- Describe IBM Integration Bus publish/subscribe functions
- Provide transformation and routing functions at publication time
- Filter publication messages based on the message content

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center for IBM Integration Bus V10



## Unit objectives

After completing this unit, you should be able to:

- Describe IBM Integration Bus publish/subscribe functions
- Provide transformation and routing functions at publication time
- Filter publication messages based on the message content

© Copyright IBM Corporation 2016

---

Figure 15-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- Introduction to publish/subscribe
- Publish/subscribe with IBM MQ
- Publish/subscribe with MQTT

© Copyright IBM Corporation 2016

Figure 15-2. Topics

WM676/ZM6761.0

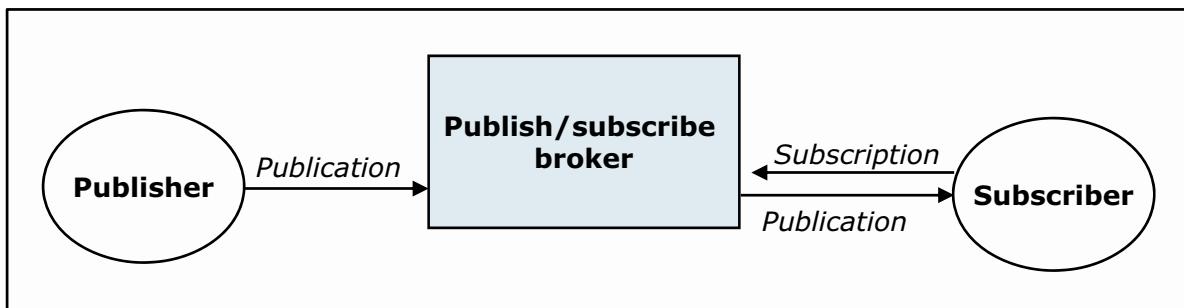
## Notes:



## **15.1.Introduction to publish/subscribe**

## What is publish/subscribe?

- Provides independence between senders (publishers) and receivers (subscribers)
  - Applications can join or leave without affecting others or configuration actions
  - Administrator can set authorities
  - If required, administrative application can set subscriptions
  - Publish/subscribe broker filters messages based on subject or topic



© Copyright IBM Corporation 2016

Figure 15-3. What is publish/subscribe?

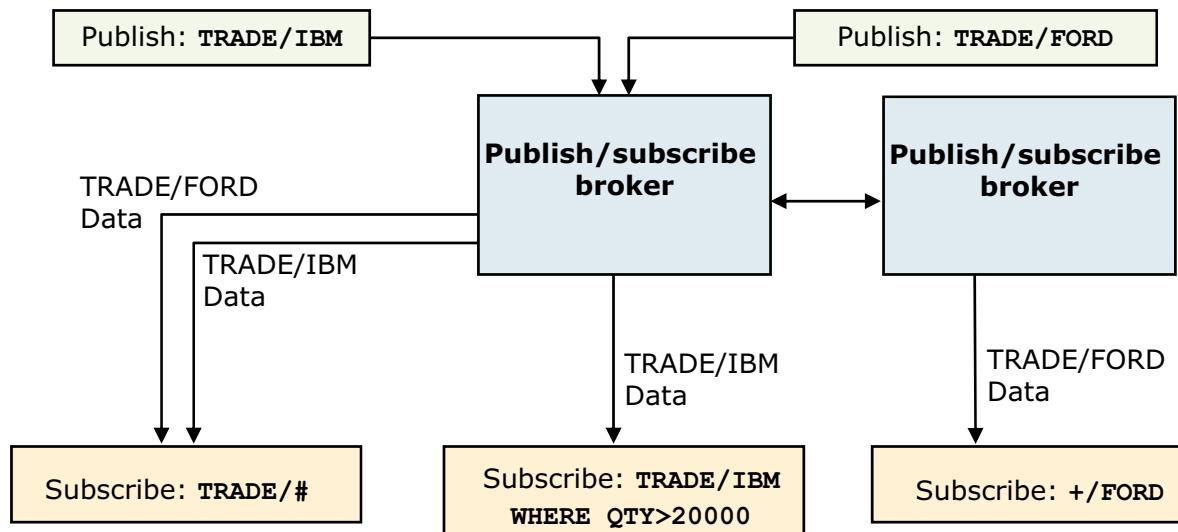
WM676/ZM6761.0

### Notes:

Publish/subscribe decouples the sender (publisher or producer) and the receiver (subscriber or consumer) of the message. Applications can register and unregister to a topic through a publish/subscribe broker such as IBM MQ.

The advantage of a publish/subscribe network is that it can remove the unmanageable aspects of a point-to-point network and replace them with a simple network of a publisher, an integration node, and all the subscribers. New subscription clients or services can be added without any effect or interruption in the service to other users. This capability provides a superior means of providing streamlined and efficient integration and growth across an enterprise.

## Topics and filters



- Topic identifies the subject matter of the publication
- Publisher can specify multiple topics for the same publication
- Subscribers can use wildcards

© Copyright IBM Corporation 2016

Figure 15-4. Topics and filters

WM676/ZM6761.0

### Notes:

In a publish/subscribe network, the topic identifier represents the subject matter of a publication, which the publisher specifies. The topic string can be any length.

A topic tree structure can be built by qualifying the topic with the slash (/) character. The slash (/) denotes a topic hierarchy. Publishers can specify multiple topics for the same publication.

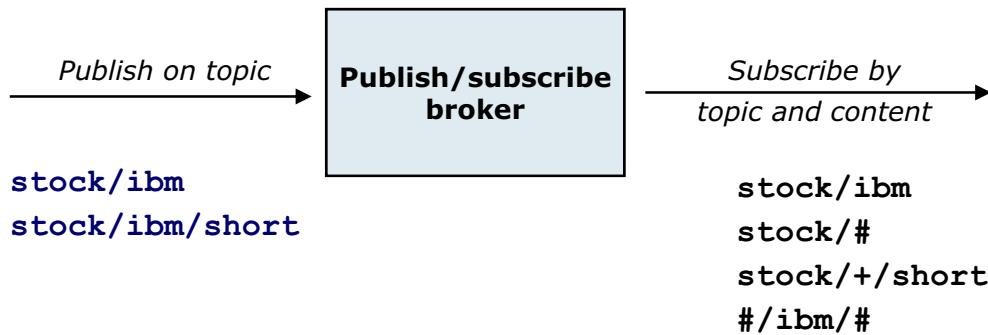
Subscribers subscribe to topics and can use wildcards:

- + matches with exactly one topic level (between / characters).
- # matches with zero or more topic levels (at the beginning or end of a topic).

Subscribers can optionally include a content filter to further refine a request for matching publications. In Integration Bus, a content filter is specified by using the same syntax as an ESQL expression that is used in a Filter node. Likewise, it is evaluated for a true or false result.

Wildcards are also allowed when specifying a content filter expression. Wildcard use conforms to SQL specifications. In this figure, the "WHERE QTY > 20000" represents a content filter.

## Topics and filters: Example



Topic = ' <code>stock/ibm</code> '	Filter = ' <code>Body.price &gt; 130</code> '
Topic = ' <code>stock/#</code> '	Filter = ' <code>Body.price*Body.volume &gt; 1000000</code> '
Topic = ' <code>#</code> '	Filter = ' <code>Body.seller like '"JPM%"'</code> '

### Key

- / topic level separator
- + single-level wildcard
- # multilevel wildcard

© Copyright IBM Corporation 2016

Figure 15-5. Topics and filters: Example

WM676/ZM6761.0

### Notes:

A subscription must include a topic. A hashtag symbol (#) denotes a generic topic.

Filters are optional. The syntax of subscription filters is identical to the syntax that is used in an Integration Bus Filter node.



Use only single quotation marks (') in filters.

The figure shows examples of valid topic and filter strings by using topic level separators, single-level wildcards, and multilevel wildcards. The plus sign (+) matches with exactly one topic level, between forward slash (/) characters. The hashtag (#) matches with zero or more topic levels, at the beginning or end of the topic.

## Topic specifications

- Topic names are case-sensitive  
For example, ACCOUNTS and Accounts are two different topics
- Topic names can include the space character
- A topic can contain one or more “/” characters; this character separates **levels** in the topic
  - Levels are used to define hierarchies within a topic
- A leading “/” creates a distinct topic  
For example, /Country is different from Country
- You can reference multiple topics and topic levels by using the wildcard characters “+” and “#”
  - “+” is the single-level wildcard character; matches exactly one topic level
  - “#” is the multi-level wildcard character; matches any number of topic levels

© Copyright IBM Corporation 2016

Figure 15-6. Topic specifications

WM676/ZM6761.0

### Notes:

You can use any characters when you specify publication and subscription topics. However, you must follow some rules, as identified in the figure.

You can use the forward slash character “/” to create a hierarchy to a topic (to define a topic “tree”). For example, you can create a topic InventoryMessage, and then create subtopics such as InventoryMessage/Sold, InventoryMessage/Restock, and InventoryMessage/ReturnToInventory. You can create more subtopics for any of these topics.

The number of levels in a topic tree has no limit, nor is there any limit to the length of the name of a level in a topic tree.

When you reference topics (for example, when registering a subscription), you can specify wildcard characters to match one or more levels of a topic string.

Be aware of the following restrictions on topic specifications:

- If the wildcard characters “+” and “#” are mixed with any other characters (including themselves but excluding the topic level separator “/”) within a topic level, they are not treated as wildcards.
- A topic name that contains “//” is an invalid name. An attempt to subscribe to a topic with such a name causes an error.

- Do not include the null character (Unicode \x0000) in any topic.
- The topic trees with roots of “\$SYS” and “\$ISYS” are reserved for use by IBM Integration Bus.

## Integration Bus and publish/subscribe support (1 of 2)

- Integration Bus enhances publish/subscribe applications
  - Provides extra transformation or routing function, or both, at publication time
  - Filters messages based on the content of the body of the message
- MQTT or IBM MQ publish/subscribe brokers notify external applications of significant events that occur in integration nodes
  - Operational events (message flow statistics, resource statistics, and workload management)
  - Administrative events (integration node status, integration server status, and integration server configuration)
  - Business events (business monitoring)
- Choose the publish/subscribe broker based on processing requirements and your existing architecture

© Copyright IBM Corporation 2016

Figure 15-7. Integration Bus and publish/subscribe support (1 of 2)

WM676/ZM6761.0

### Notes:

Integration Bus supports both IBM MQ and MQTT publish/subscribe brokers.

MQTT is a lightweight publish/subscribe messaging protocol. You can use Integration Bus to connect to applications and devices that send and receive messages by using the MQTT messaging protocol.

You can use publish/subscribe in IBM Integration Bus for these two situations:

- To provide more transformation or routing function, or both, at publication time.
- To filter messages based on the content of the body of the message.

As shown in the figure, messages are supplied to the message flow by applications that publish messages. These applications are referred to as *publishers*.

Messages are retrieved from the message flow by applications that registered a subscription with an integration node. Those applications are referred to as *subscribers*. A *subscription* defines the interest that a subscriber has in published messages.

## Integration Bus and publish/subscribe support (2 of 2)

- Integration Bus contains a built-in MQTT broker that is enabled by default for all events that the integration node generates except for business events
- If IBM MQ is installed, you can use the IBM MQ publish/subscribe broker in place of, or with the built-in MQTT broker
  - If an IBM MQ queue manager is specified on the integration node, the IBM MQ publish/subscribe broker is enabled by default for the integration node events
- Modify the configuration of the publish/subscribe broker by using the `mqsichangeproperties` command
 

Example: `mqsichangeproperties IBNODE -b pubsub  
-o BusinessEvents/MQTT -n enabled -v true`
- Integration Bus built-in nodes for publish/subscribe
  - MQTT Subscribe
  - MQTT Publish
  - Publication

© Copyright IBM Corporation 2016

Figure 15-8. Integration Bus and publish/subscribe support (2 of 2)

WM676/ZM6761.0

### Notes:

IBM Integration Bus supports two types of publish/subscribe broker: MQTT and IBM MQ. You can choose the type of publish/subscribe broker to use based on your processing requirements and your existing architecture.

A built-in MQTT broker is provided with Integration Bus, and MQTT publication is enabled by default for all events that the integration node generates, except for business events. If you choose not to use the built-in MQTT broker, you can specify an alternative MQTT broker.

If you installed IBM MQ, you can use the IBM MQ publish/subscribe broker.

You can modify the configuration or disable the MQTT publish/subscribe broker by using the `mqsichangeproperties` command.

You can create message flows to receive an MQTT message by using the MQTT Subscribe node to subscribe to one or more topics on an MQTT server. You can send an MQTT message by using the MQTT Publish node in your message flow to publish messages to a topic on an MQTT server.

## Viewing publish/subscribe properties

- Use the `mqsi reportproperties` command with `-b pubsub` to examine the values of the publish/subscribe properties on an integration node

Example:

```
mqsi reportproperties IBNODE -b pubsub -o AllReportableEntityNames -r

OperationalEvents
MQ
    policyUrl=''
    enabled='true'
    format=''

MQTT
    policyUrl=''
    enabled='true'

AdminEvents
MQ
    policyUrl=''
    enabled='true'
    format=''

MQTT
    policyUrl=''
```

© Copyright IBM Corporation 2016

Figure 15-9. Viewing publish/subscribe properties

WM676/ZM6761.0

### Notes:

You can use the `mqsi reportproperties` command with the `-b pubsub` option to determine whether the IBM MQ or the built-in MQTT publish/subscribe broker is enabled for an integration node.



## 15.2.Publish/subscribe with IBM MQ



## IBM MQ publish/subscribe

- IBM MQ queue manager implements publish/subscribe
  - Retains subscriptions and publications as necessary
  - Matches publications to subscriptions
  - Sends subscriptions/publications to other integration nodes to minimize network traffic
  - Supports direct communication with several integration nodes to the same queue manager
- Can manage subscriptions and topics by using IBM MQ Explorer

© Copyright IBM Corporation 2016

Figure 15-10. IBM MQ publish/subscribe

WM676/ZM6761.0

### Notes:

You can use IBM MQ to support publish/subscribe applications.

The figure lists some of the publish/subscribe tasks that can be done with IBM MQ.

IBM MQ publish/subscribe applications typically send free-form messages with specific publish/subscribe commands in an MQRFH or MQRFH2 header that are directed to appropriate integration node queues.



## Managing publish/subscribe with IBM MQ Explorer

The screenshot shows two instances of the IBM WebSphere MQ Explorer interface. The top instance displays the 'Topics' view, which lists a single topic named 'Sports' with a description of 'Football/Team'. A context menu is open over this topic, listing options like 'Compare with...', 'Delete...', 'Status...', and several testing and configuration commands. The bottom instance displays the 'Subscriptions' view, which lists three subscriptions: 'Football' (Topic name 'Sports', Topic string 'Football/Teams'), 'IIBQM SYSTEM.BROKER.INTER.BROKER.COMMUNICATION...' (Topic name 'SYSTEM.BROKER....', Topic string 'SYSTEM.BROKER.ADMIN.STREAM/MQ/IIBQM'), and 'SYSTEM.DEFAULT.SUB' (Topic name 'SYSTEM.DEFAULT.SUB', Topic string 'SYSTEM.DEFAULT.SUB'). Both instances have the 'MQ Explorer - Navigator' view open on the left, showing the queue manager structure.

**Topics** view lists the properties of all topics on the queue manager

**Subscriptions** view lists the properties of all subscriptions on the queue manager

© Copyright IBM Corporation 2016

Figure 15-11. Managing publish/subscribe with IBM MQ Explorer

WM676/ZM6761.0

### Notes:

You can use IBM MQ Explorer or IBM MQ commands to manage publication topics.

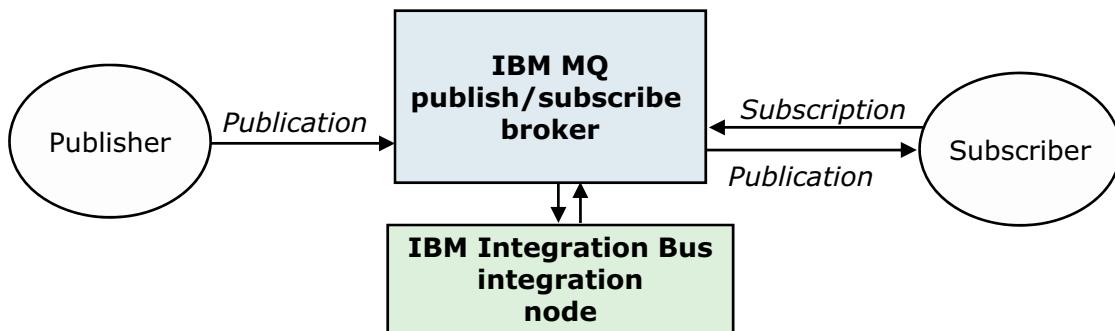
To access the topics on a queue manager, click the **Topics** folder under the queue manager name in the **MQ Explorer – Navigator** view. From the **Topics** view, you can create new topics and check topic status. You can also test publications.

You can also use IBM MQ Explorer or IBM MQ commands to manage subscriptions.

To access the subscriptions on a queue manager, click the **Subscriptions** folder under the queue manager name in the **MQ Explorer – Navigator** view. From the **Subscriptions** view, you can create new subscriptions and check subscription status. You can also test subscriptions.

## IBM Integration Bus and IBM MQ

- Integration Bus enhances IBM MQ publish/subscribe applications
  - Provides extra transformation or routing, or both, at publication time
  - Extends the message selection support that IBM MQ provides by using EQSL expressions to filter messages based on the message content
- Requirements:
  - Integration node must specify IBM MQ queue manager
  - IBM MQ queue manager must contain Integration Bus SYSTEM.BROKER queues



© Copyright IBM Corporation 2016

Figure 15-12. IBM Integration Bus and IBM MQ

WM676/ZM6761.0

### Notes:

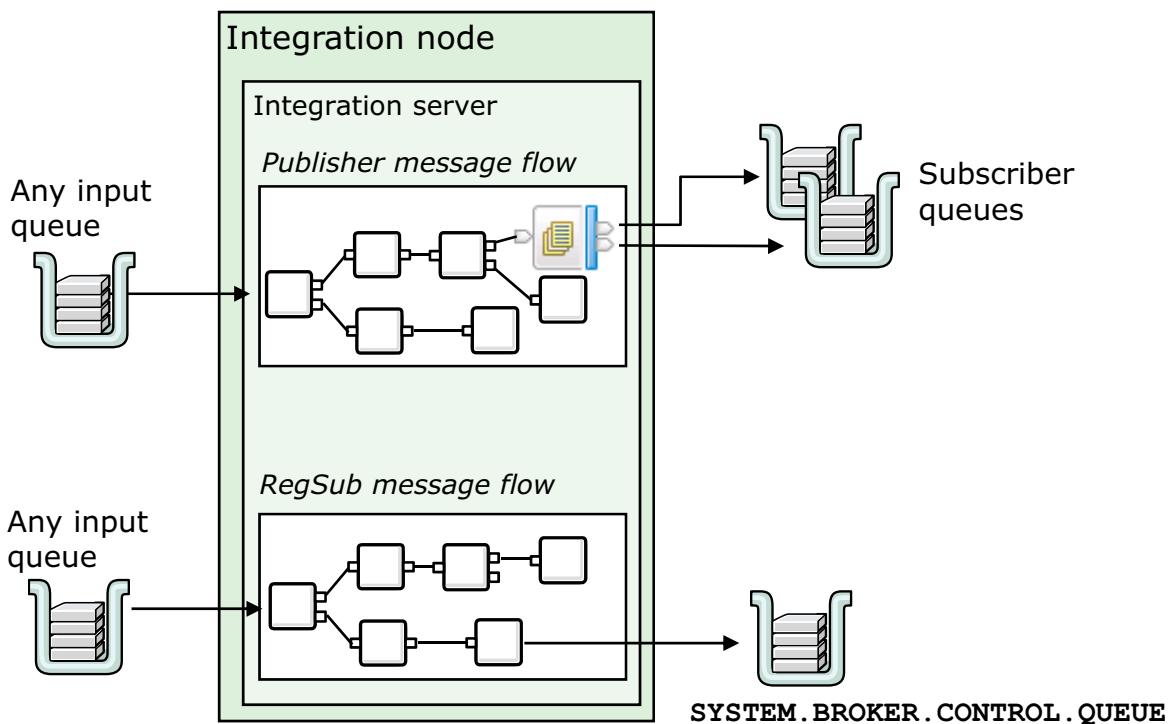
IBM MQ can specify a filter when a subscription is made, but the filter can refer to items in headers only. Integration Bus enhances IBM MQ publish/subscribe applications by acting as a content filter provider for IBM MQ. Integration Bus allows subscriptions to specify extended filters that can refer to elements in the body of publications.

Integration Bus can also be used to provide extra transformation and routing for publications.

The requirements for IBM MQ content filtering are listed in the figure.

IBM MQ publish/subscribe for distributed operating systems is taught in course WM212, *IBM MQ V8 Advanced System Administration (Distributed)*.

## Publish/subscribe message flows and applications



© Copyright IBM Corporation 2016

Figure 15-13. Publish/subscribe message flows and applications

WM676/ZM6761.0

### Notes:

Message flows can complete publish/subscribe tasks for applications. This approach allows existing applications to publish messages or register as a subscriber without being modified with publish/subscribe commands.

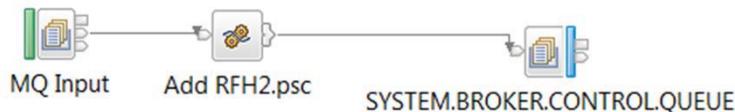
In the top example, a publishing application publishes messages by using a Publication node. Those messages are sent to the subscriber queues by Integration Bus and IBM MQ.

In the bottom example, a Register Subscription (RegSub) message flow registers subscriptions by reading from an input queue and then writing a Register Subscription message to the SYSTEM.BROKER.CONTROL.QUEUE. A more detailed example of this technique is shown on the next pages.

## Registering subscriptions in IBM Integration Bus (1 of 2)

You can register subscriptions in a number of different ways:

- Use a message flow to send application message that contains RegSub command to SYSTEM.BROKER.CONTROL.QUEUE



- Use IBM MQ SupportPac IH03, RfhUtil, to register static subscriptions manually

© Copyright IBM Corporation 2016

Figure 15-14. Registering subscriptions in IBM Integration Bus (1 of 2)

WM676/ZM6761.0

### Notes:

Integration Bus has no specific “Subscribe” message processing node for IBM MQ. You can build a message flow that receives messages on any input queue, build an MQRFH2 header that contains the register subscriber command, and send this message to SYSTEM.BROKER.CONTROL.QUEUE.

Details on how to format the MQRFH2 header are shown in the next page.

The alternative is to use a tool such as RFHUtil to build a subscription statically. You should be aware that static registration means that the Integration Bus administrators must take responsibility for maintaining and deleting subscriptions for subscriber applications. Managing subscriptions in this way does not take advantage of the dynamic nature of subscriptions.

## Registering subscription in IBM Integration Bus (2 of 2)

- Use Compute node to set the MQRFH2 header

```
SET OutputRoot.MQRFH2.psc.Command = 'RegSub';
SET OutputRoot.MQRFH2.psc.Topic = 'Complaint/Order/#';
SET OutputRoot.MQRFH2.psc.Filter = 'Body.Complaint.Text like
''%late%'''';
SET OutputRoot.MQRFH2.psc.QName = 'MYQUEUE';
SET OutputRoot.MQRFH2.psc.QMgrName = 'MYQM';
```

- Integration Bus replies are sent to `MQMD.ReplyToQueue` if the subscription was a request message
- Queue manager automatically deletes subscriptions for the following cases:
  - Subscription expires (`MQMD.Expiry` in RegisterSubscriber message)
  - Temporary (local) delivery queue deleted
  - Subscriber application ends

© Copyright IBM Corporation 2016

Figure 15-15. Registering subscription in IBM Integration Bus (2 of 2)

WM676/ZM6761.0

### Notes:

The MQRFH2 folder contains a number of folders that applications that communicate with IBM MQ use. These folders include:

- Message content descriptor (`mcd`)
- Publish/subscribe command (`psc`)
- Publish/subscribe command reply (`pscr`)
- User-defined properties (`usr`)
- Java Messaging Service (`jms`)

To programmatically register a subscription, you can use Compute or JavaCompute nodes to populate the `MQRFH2.psc` header, and then send the message to IBM MQ. When the integration node receives a `RegSub` command from `SYSTEM.BROKER.CONTROL.QUEUE`, it first checks the validity of the subscription.

It is not possible to subscribe with a subscription queue that does not exist. So either a local queue (or a local definition of a remote queue) must exist on the integration node queue manager for that queue that is identified in the `psc.QName` attribute. The integration node cannot determine whether the definitions on the remote queue manager are valid.

If no `psc.QName` is specified in the `MQRFH2` header, the `MQMD.ReplyToQueue` is used as subscriber queue instead.

The integration node also checks subscription conditions to ensure that no registered subscription results in the same publication that is delivered to any subscriber queue twice.

You can check the success of the `RegSub` command by using the normal IBM MQ request/reply mechanism. Integration Bus sends a reply message that consists of a `MQRFH2.pscr` folder to the reply-to queue that is specified in the `MQMD` of the subscription message.

You can send a *Delete Publication* request in a similar way.

## Publication node

- Publishes a message through the IBM MQ queue manager that is specified on the integration node
- Filters output messages from a message flow and transmits them through an IBM MQ publish/subscribe broker to subscribers
- Only one copy of a message is sent to each subscriber queue, regardless of how many subscriptions match
- Uses the topic from MQ Input node
  - Can override in `Properties.Topic` or `MQRFH2.psc.Topic` by using a compute-type node

```
SET OutputRoot.MQRFH2.psc.Command = 'Publish';
SET OutputRoot.MQRFH2.psc.Topic =
CASE InputBody.Message.Complaint.Type
  WHEN 'Order' THEN 'Complaint/Order'
  WHEN 'Delivery' THEN 'Complaint/Delivery'
  ELSE 'Complaint/Other'
END;
```

© Copyright IBM Corporation 2016

Figure 15-16. Publication node

WM676/ZM6761.0

### Notes:

You can identify a subscription point within the properties of the Publication node. The subscription point differentiates multiple Publication nodes in the same message flow, which represents a specific path through the message flow, for example, if the same message is published in different formats. Subscribers can choose the subscription point to which they subscribe.

An unnamed Publication node (that is, one without a specific subscription point) is known as the default Publication node.

The Publication node always tries to deliver its publication. If the subscriber queue (or in the case of a remote subscriber queue, the transmission queue) is not reachable, the integration node attempts to put the message to the dead-letter queue. If it is not possible to use the dead-letter queue and the publication flow is transactional, all publications are rolled back.

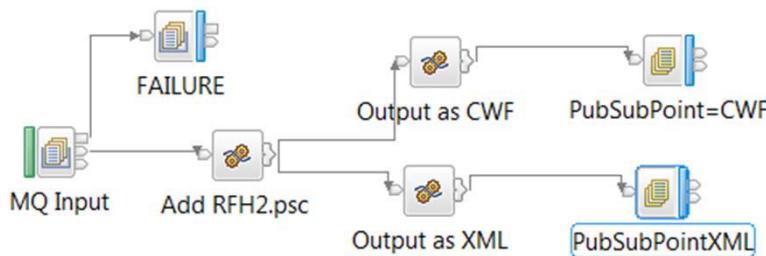
Having zero registered subscribers does not cause an error. The incoming message is propagated through the **NoMatch** terminal of the Publication node.

## Publication flows

- Simple flow example: No coding, no customization options, no RFH2 header in publication



- Customized flow example: Set all options and topics in `MQRFH2.psc` (MQRFH2 header in publication)



© Copyright IBM Corporation 2016

Figure 15-17. Publication flows

WM676/ZM6761.0

### Notes:

In the publication flow, the Integration Bus Publication node sends the publication message to the `SYSTEM.BROKER.CONTROL.QUEUE`. Publication nodes are used instead of MQ Output nodes.

The Publication node has an **Out** terminal and a **NoMatch** terminal. The Publication node does not have a **Failure** terminal (the integration node answers on the `ReplyToQueue` regardless of success or failure).

The message that is being published must be a *Publish* command message, a *Delete Publication* command message, or have at least one topic present in the standard properties of the message.

The Publication node uses the topic or topics, and any options present in the command message, to publish the message.

A topic must be specified. Even if the message does not contain `MQRFH2.psc` commands, it is possible to specify a default topic in the MQ Input node. In that case, the publication does not contain an RFH2 header, which might be needed for receiver applications that cannot handle this header. An alternative is to use one of the compute-type nodes to set the `OutputRoot.Properties.Topic` field. Doing so has the same effect, plus the additional advantage of being able to set the topic dynamically.

To specify further publish/subscribe options, such as retained publication, local publication, and others, you must set up the appropriate publish/subscribe commands in the MQRFH2 header. If the sending application does not supply an MQRFH2 header, use a compute-type node in the message flow to add the MQRFH2 publish/subscribe commands.

## Content-based filtering

- Integration Bus can act as a content filtering provider for IBM MQ to allow extended filters to message body based on subscribers selection string
  - Uses ESQL expressions to filter on the entire message by using the Root, Body, and Properties correlation names
  - Accesses the message tree as read-only
- Integration Bus provides content filtering services for IBM MQ subscribers:
  - MQRFH2
  - MQSUB
- Service runs within nominated integration servers
  - If Integration Bus does not have at least one integration server that is enabled for content-based filtering, IBM MQ can support only “message selection”

© Copyright IBM Corporation 2016

Figure 15-18. Content-based filtering

WM676/ZM6761.0

### Notes:

IBM MQ supports a limited amount of filtering on the message header and message properties, but it cannot filter on the body of the message because it typically cannot parse this part of the message. Because IBM Integration Bus can parse the message body, it extends the IBM MQ filtering capability.

Content-based filtering allows a subscriber to restrict or filter messages that it wants to receive, in addition to the topics that it specifies. This filter is specified as an SQL expression, for example:  
Body.Name LIKE 'Smit%'

## Content-based filtering configuration

- Enabled by using the `mqsischangeproperties` command with the `-o ContentBasedFiltering` and `-n cbfEnabled` properties

Example:

```
mqsischangeproperties IBNODE -e ISPUBSUB
-o ContentBasedFiltering -n cbfEnabled -v true
```

- On z/OS
  - Verify that the integration node started task ID has UPDATE access permission to profile `MQ_QMNAME.BATCH` of class MQCONN
  - If you enable content-based filtering in multiple integration servers, it is active in only one integration server at any time
- On Linux, UNIX, Windows, grant authorization for the system to the queue manager with the IBM MQ `setmqaut` command:

```
setmqaut -t Qmanager +system -p IntNodeUserId
```

- Restart the integration server for the change to take effect

© Copyright IBM Corporation 2016

Figure 15-19. Content-based filtering configuration

WM676/ZM6761.0

### Notes:

Content-based filters in Integration Bus extend IBM MQ message selectors to allow the use of ESQL expressions to filter on the entire message. Content-based filtering must be enabled for any integration servers where you want content-based filtering to be active.

You can enable content-based filtering on an integration server by using the IBM Integration web interface or by using the `mqsischangeproperties` command.

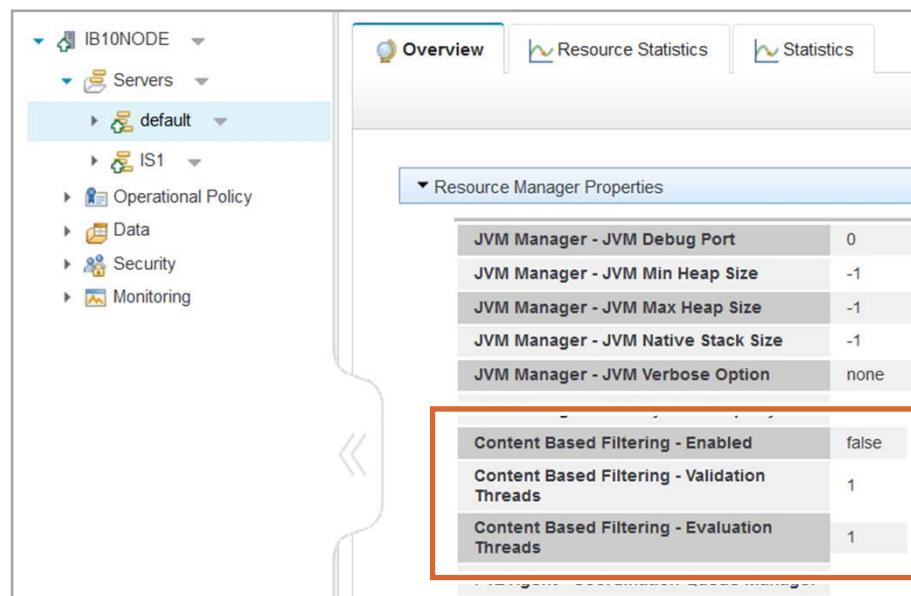
You enable content-based filtering on an integration server by using the `mqsischangeproperties` command with the `-o ContentBasedFiltering -n cbfEnabled -v true` parameters, as shown in the figure.

On distributed systems, you must also grant authorization to the queue manager by using the IBM MQ `setmqaut` command.

## Configure content-based filtering in Integration web interface

- Current settings are shown in the integration server **Resource Manager Properties**

- Click **Edit** to change the values
- Restart the integration server for the changes to take effect



© Copyright IBM Corporation 2016

Figure 15-20. Configure content-based filtering in Integration web interface

WM676/ZM6761.0

### Notes:

You can view and modify the content-based filtering properties in the Integration web interface by clicking an integration server and then expanding the **Resource Manager Properties** on the **Overview** tab.

You must restart the integration server for the change to take effect.

## 15.3.Publish/subscribe with MQTT

## MQTT overview

- MQTT is a messaging protocol that provides robust messaging features for communicating with remote systems and devices, and also minimizes network bandwidth and device resource requirements
- Supports always-connected and sometimes-connected models
- Provides multiple message delivery qualities of service:
  - 0 = message is delivered at most one time
  - 1 = message is delivered but might be duplicated
  - 2 = one time only delivery
- MQTT client is integrated with IBM MQ as a publish/subscribe application
- IBM MQ publish/subscribe broker manages the topics and subscriptions that MQTT clients create

© Copyright IBM Corporation 2016

Figure 15-21. MQTT overview

WM676/ZM6761.0

### Notes:

MQTT is a protocol that is designed for devices in constrained environments, such as embedded systems, cell phones, and sensors with limited processing ability and memory, and for systems that are connected to unreliable networks.

IBM MQ can be used to manage MQTT topics and subscriptions.



## MQTT administration with IBM MQ Explorer

- MQTT subscriptions appear as IBM MQ subscriptions
  - Subscription name is: `mqtt_MQTTClientIdentifier_Topic`
  - Can be administered and deleted from **Subscriptions** view
- MQTT and IBM MQ share the topic space
  - Publish from MQTT to IBM MQ and vice versa
- IBM MQ provides a sample MQTT configuration through IBM MQ Explorer when you install the **Telemetry** option

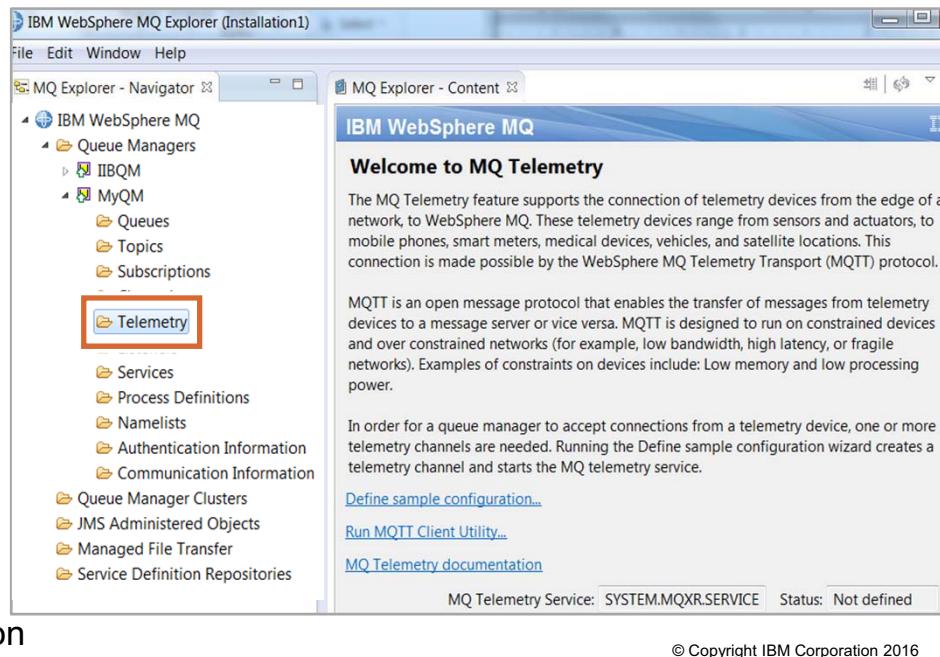


Figure 15-22. MQTT administration with IBM MQ Explorer

WM676/ZM6761.0

### Notes:

You can use IBM MQ Explorer to administer MQTT subscriptions and topics when you install the **Telemetry** option.

IBM MQ Explorer also provides a sample MQTT configuration and client.



## MQTT support in IBM Integration Bus

- Integration Bus provides built-in message processing nodes for processing MQTT messages
- MQTT broker that is provided with Integration Bus provides publish/subscribe for Integration Bus events
  - Enabled by default for administrative and operational events
- Events can be routed to IBM MQ (when a queue manager is associated with an integration node) or an MQTT broker
  - MQTT publications can be routed to an external MQTT broker such as IBM MessageSight
  - Events are published to the IBM MQ queue manager associated with the integration node for compatibility with an earlier version

© Copyright IBM Corporation 2016

Figure 15-23. MQTT support in IBM Integration Bus

WM676/ZM6761.0

### Notes:

MQTT Subscribe node to subscribe to one or more topics on an MQTT server. You can send an MQTT message by using the MQTT Publish node in your message flow to publish messages to a topic on an MQTT server.

Integration Bus publishes status and event information in known topics. You can use the Integration Bus built-in MQTT broker for events that integration servers emit.

## Configuring the Integration Bus MQTT broker

- Modify the configuration by using the `mqsischangeproperties` command with the `MQTTServer` object and `pubsub` component
  - Use the `enabled` property to enable or disable the built-in MQTT broker (default is `true`)
  - Use the `port` property to specify the port that the broker uses (default is 11883)
- View the current configuration by using the `mqsireportproperties` command with the `MQTTServer` object and `pubsub` component

Example: View the port that the MQTT broker uses

```
mqsireportproperties IBNODE -b pubsub -o MQTTServer -n port
```

- Use the `enabled` property of the `mqsischangeproperties` command to enable or disable the publication of event messages by a specified publish/subscribe broker (`OperationalEvents`, `AdminEvents`, and `BusinessEvents`)

Example: `mqsischangeproperties IBNODE -b pubsub -o BusinessEvents/MQTT -n enabled -v true`

© Copyright IBM Corporation 2016

Figure 15-24. Configuring the Integration Bus MQTT broker

WM676/ZM6761.0

### Notes:

You can modify the configuration of the built-in MQTT broker by using the `mqsischangeproperties` command.

Use the `enabled` property for the `-o MQTTServer` object in the `-b pubsub` component to enable or disable the built-in MQTT broker. Use the `-n port` property to specify the port to be used by the broker. By default, the `enabled` property is set to `true` and the port is set to 11883.

When an integration node starts, the built-in MQTT broker starts on the port that the MQTT server port property specifies. If more than one integration node is configured with the same MQTT server port, only one MQTT broker starts. All integration nodes that use the same MQTT server port, use the same MQTT broker to publish their events.

Subscribers that connect to the MQTT broker receive all the events that the broker publishes, unless the subscriber includes the name of the integration node in their subscriptions.

You can view the current configuration of the built-in MQTT broker by using the `mqsireportproperties` command.

## Support for MQTT in a message flow

- MQTT Subscribe node
  - Receives messages from an application or device that publishes messages by using the MQTT messaging protocol
  - Stores information about the message in `LocalEnvironment.MQTT.Inputs`
- MQTT Publish node
  - Publishes messages from a message flow to a topic on an MQTT server
  - You can dynamically override some node properties with elements in the `LocalEnvironment` message tree
- Can attach operational policies to MQTT nodes to specify connection information

© Copyright IBM Corporation 2016

Figure 15-25. Support for MQTT in a message flow

WM676/ZM6761.0

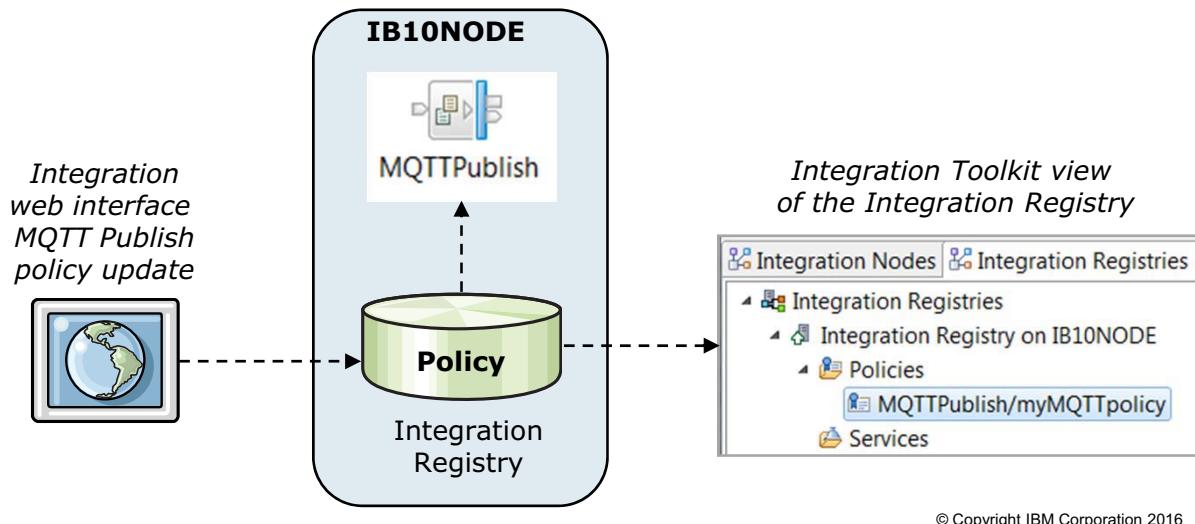
### Notes:

A developer can use the MQTT Publish node to publish messages from a message flow to a topic on an MQTT server. When you use the MQTT Publish node in a message flow, you can dynamically override some of its properties with elements in the local environment message tree.

A developer can use the MQTT Subscribe node to receive messages from an application or device that publishes messages by using the MQTT messaging protocol. Integration Bus can then propagate these messages in a message flow.

## MQTT operational policies

- Manage by using the Integration web interface, Integration Toolkit, or `mqsicreatepolicy` command
- Stored in Integration Registry on the integration node
- Can update at run time without redeployment of an application or integration service



© Copyright IBM Corporation 2016

Figure 15-26. MQTT operational policies

WM676/ZM6761.0

### Notes:

Administrators and developers can use operational policies to control the behavior of message flows, and the nodes within message flows, at run time without redeploying resources.

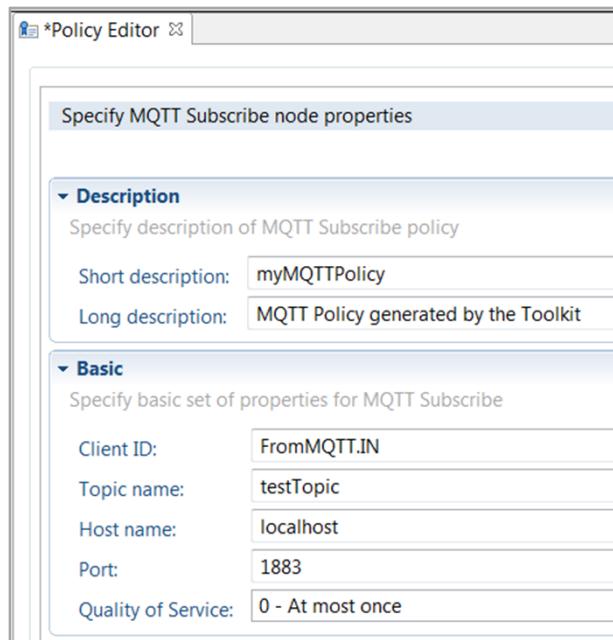
- An MQTT Publish policy can be attached to one or more MQTT Publish nodes in a message flow to control the value of specific MQTT publishing properties at run time.
- An MQTT Subscribe policy can be attached to one or more MQTT Subscribe nodes in a message flow to control the value of specific MQTT subscription properties at run time.

Policies can be created and updated at any time in the solution lifecycle. A policy instance is made up of a set of defined operational properties that are stored in a policy document, and a policy instance can be applied at the message flow, or node level. You can set operational property values on the node at development time, override them in the BAR file, and set them in an operational policy.



## Creating an MQTT policy in the Integration Toolkit

1. In the Message Flow editor, select the MQTT node that you want to use to generate a policy document.
2. In the **Properties** view for this node, click the **Policy** tab.
3. Click **Generate new policy** to open the Policy editor.
4. Edit the values of the operational properties as required.
5. Click **Save**, enter a policy name, select the target integration node, and click **Finish** to store the policy document in the Integration Registry.



© Copyright IBM Corporation 2016

Figure 15-27. Creating an MQTT policy in the Integration Toolkit

WM676/ZM6761.0

### Notes:

As shown in the figure, you can define an MQTT policy in the Integration Toolkit.

By default, the generated policy is attached to this message flow node at run time. To save the policy without attaching it to the node, clear the **Attach the generated policy to the node** check box.



## Updating an attached MQTT policy

- Use the Integration web interface to retrieve and update a policy that is attached to a message flow node
  1. Expand **Servers > IntServer > Application**, where *IntServer* is the name of your integration server, and *Application* is where you stored your message flow.
  2. Expand **Message Flows**, and select the name of the message flow, or subflow to view.
  3. Select the **Operational Policy** tab from the top of the message flow pane to display the message flow or subflow in the **Node Policies** section.
  4. Click the **Policy** icon to retrieve and update the policy document.

© Copyright IBM Corporation 2016

Figure 15-28. Updating an attached MQTT policy

WM676/ZM6761.0

### Notes:

You can also use the message flow view to retrieve and update a policy that is attached to a message flow node.

1. In the navigation tree, expand **Servers > IntServer > Application**, where *IntServer* is the name of your integration server, and *Application* is where you stored your message flow.
2. Expand **Message Flows**, and select the name of the message flow, or subflow, you want to view.
3. Select the **Operational Policy** tab from the top of the message flow pane, and the message flow, or subflow, is displayed in the **Node Policies** section.
4. If the message flow, or subflow, includes a node that has an operational policy that is attached, the policy icon is displayed on the upper-right corner of the node icon. Click the **Policy** icon to retrieve and update the policy document.



## Unit summary

Having completed this unit, you should be able to:

- Describe IBM Integration Bus publish/subscribe functions
- Provide transformation and routing functions at publication time
- Filter publication messages based on the message content

© Copyright IBM Corporation 2016

---

Figure 15-29. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. Which of the following options is an expression that is applied to the content of a publication message to determine whether it matches a subscription?
  - a. Filter
  - b. Exception
  - c. Message descriptor
2. True or False: The **NoMatch** terminal on the Publication node propagates a message if there are no subscribers for the topic that is associated with the message.
3. List two advantages of the publish/subscribe messaging model that is compared with the point-to-point messaging model.

© Copyright IBM Corporation 2016

Figure 15-30. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.

## Checkpoint answers

1. Which of the following options is an expression that is applied to the content of a publication message to determine whether it matches a subscription?
  - a. Filter
  - b. Exception
  - c. Message descriptor

Answer: a

2. True or False: The **NoMatch** terminal on the Publication node propagates a message if there are no subscribers for the topic that is associated with the message.

Answer: True

3. List two advantages of publish/subscribe messaging model that is compared with the point-to-point messaging model.

Answer: Publisher and subscriber are decoupled. Publishers and subscribers can be added and removed without any effect on the others.

© Copyright IBM Corporation 2016

---

Figure 15-31. Checkpoint answers

WM676/ZM6761.0

### Notes:

# Unit 16. Monitoring message flow events

## What this unit is about

Integration nodes use a publish/subscribe broker to publish events in response to changes in configuration, state, or operational status. You can monitor these events by subscribing to the topics. For audit or problem determination purposes, you can record data to a database and then view it and replay it. This unit describes the IBM Integration Bus tools that are available for monitoring message flow events and analyzing message data.

## What you should be able to do

After completing this unit, you should be able to:

- Define monitoring events in the message flow
- Use the record and replay function to capture and review processed messages

## How you will check your progress

- Checkpoint
- Lab exercise

## References

IBM Knowledge Center for IBM Integration Bus V10

## Unit objectives

After completing this unit, you should be able to:

- Define monitoring events in the message flow
- Use the record and replay function to capture and review processed messages

© Copyright IBM Corporation 2016

---

Figure 16-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Topics

- Monitoring events
- Record and replay

© Copyright IBM Corporation 2016

Figure 16-2. Topics

WM676/ZM6761.0

## Notes:



## 16.1. Monitoring events

## Event monitoring and auditing

- Generates monitoring and audit events from message flows
  - Operationally enable, disable, and change event production in the Integration Toolkit Message Flow editor or by using commands
  - Events are published on well-known topics over IBM MQ transport or MQTT for multiple concurrent consumers
  - Events are optionally produced within same transaction sync point for optimum performance
- Provides noninvasive event monitoring profile
  - Configurable service overrides the monitoring properties of a message flow
  - Can be used to customize events without deploying the message flow again
- Integrates with IBM Business Monitor
  - Monitor and analyze key performance indicators
  - Automatic generation of monitor model
  - Comprehensive sample is built in

© Copyright IBM Corporation 2016

Figure 16-3. Event monitoring and auditing

WM676/ZM6761.0

### Notes:

Events are published over a publish/subscribe network on a well-defined topic. The topic name includes the name of the integration node, integration server, and the name that is given to the event as it is generated. It is also possible to generate an event at design time or operationally at any point in the flow. The event can contain the entire payload or any piece of it.

A message-driven bean captures the publication of business events and pushes them into a common event infrastructure for integration with IBM Business Monitor. Business Monitor can be used to trend the data. For example, Business Monitor can examine volume changes, or sort the information by geographical area.

## Event message categories

- Operational events
  - Message flow statistics
  - Resource statistics
  - Workload management
- Administrative events
  - Integration node status
  - Integration server status
  - Integration server configuration
- Business events
  - Business (message flow) monitoring

© Copyright IBM Corporation 2016

Figure 16-4. Event message categories

WM676/ZM6761.0

### Notes:

IBM Integration Bus event messages are categorized as operational events, administrative events, and business events. This unit focuses on the business events that a message flow can generate.

If you want to receive event messages for any of these events (which include viewing statistics in the IBM Integration web user interface), you must ensure that event publication is enabled. You also must ensure that a publish/subscribe broker is configured.

## Configuring the publication of event messages

- Integration node event messages can be published to the following publish/subscribe brokers:
  - Built-in MQTT publish/subscribe broker
  - MQTT publish/subscribe broker on an external MQTT server
  - IBM MQ publish/subscribe broker
  - Combination of IBM MQ and MQTT publish/subscribe brokers
- Default publish/subscribe broker depends on the Integration Bus deployment:
  - If IBM MQ is not installed, operational and administrative event messages are published to the built-in MQTT publish/subscribe broker
  - If IBM MQ is installed but a queue manager is not specified on the integration node, operational and administrative event messages are published to the built-in MQTT publish/subscribe broker
  - If IBM MQ is installed and a queue manager is specified on the integration node, operational, administrative, and business event messages are published to the IBM MQ publish/subscribe broker

© Copyright IBM Corporation 2016

Figure 16-5. Configuring the publication of event messages

WM676/ZM6761.0

### Notes:

You can configure the publication of event messages, including whether messages are published for an event message group. You can also define the publish/subscribe broker to which the messages are published. The topic is named with the integration node name, integration server, and the name that is given to the event as it is generated.

Integration node event messages can be published to the following publish/subscribe brokers:

- The built-in MQTT publish/subscribe broker that IBM Integration Bus provides
- An MQTT publish/subscribe broker on an external MQTT server
- An IBM MQ publish/subscribe broker
- A combination of IBM MQ and MQTT publish/subscribe brokers

The default publish/subscribe broker that is used for each group of event messages depends on your IBM Integration Bus deployment. By default, business events are not published to the MQTT publish/subscribe broker.

If IBM MQ is not installed, or IBM MQ is installed but a queue manager is not specified on the integration node, the messages are published to the following locations by default:

- Operational events and administration events are published to the built-in MQTT publish/subscribe broker.
- Business events are not published. If you want to publish business events, you must enable the publication of the BusinessEvents group to the built-in MQTT publish/subscribe broker.

If IBM MQ is installed and a queue manager is specified on the integration node, all events are published to the IBM MQ publish/subscribe broker by default.

## Subscribing to business event messages

- To receive business event messages from message flows, subscribe to the following topic:
 

`topicRoot/Monitoring/IntServer/messageFlow`

  - For an MQTT publish/subscribe broker, `topicRoot` is `IBM/IntegrationBus`
  - For an IBM MQ publish/subscribe broker, `topicRoot` is `$SYS/Broker`
- Use the `enabled` property of the `mqsichangeproperties` command to enable or disable the publication of event messages by a specified publish/subscribe broker (OperationalEvents, AdminEvents, and BusinessEvents)

Example:

`mqsichangeproperties IBNODE -b pubsub  
-o BusinessEvents/MQTT -n enabled -v true`

© Copyright IBM Corporation 2016

Figure 16-6. Subscribing to business event messages

WM676/ZM6761.0

### Notes:

You can subscribe to topics that return messages about the configuration, state, or operational status of your integration node, integration server, or message flows.

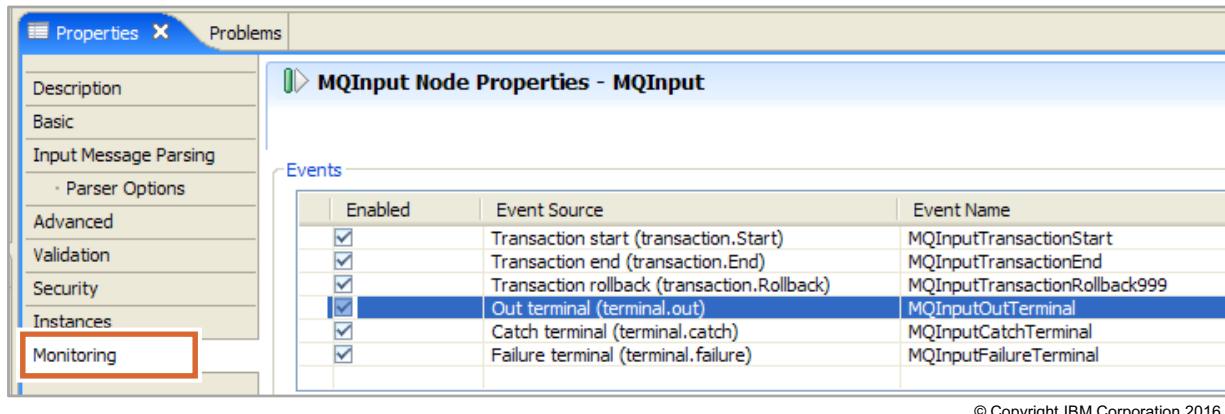
You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

When you use the built-in MQTT publish/subscribe broker, monitoring events are not published by default. You must enable the publication of monitoring events to the built-in MQTT publish/subscribe broker.

## Message flow event monitoring configuration

- Every message flow node has a **Monitoring** tab to enable monitoring events
  - Transaction start, end, and rollback events
  - An event whenever a message is propagated from any terminal of any node
- Optionally, configure payload data, content style, identity, correlation, and sequencing data
- Configure event filters to specify conditions for event creation

Example: Generate event when `msg.Price > 100`



© Copyright IBM Corporation 2016

Figure 16-7. Message flow event monitoring configuration

WM676/ZM6761.0

### Notes:

In a development environment, the monitoring requirements are defined in the Integration Toolkit on the **Monitoring** tab on the message flow node properties.

Developers can use the **Monitoring** properties to declare the event to see from a node, such as an input event, by checking the box next to the event. In the example, the MQ Input node has monitoring enabled at all points for a transaction: transaction start, transaction end, transaction rollback, **Out** terminal, **Catch** terminal, and **Failure** terminal.

It is also possible to declare sequencing information, correlating information, what the payload looks like, certain fields, and other information.

Event filters can be defined to generate events only when a certain condition applies, such as when a field in the message exceeds a specified value.

## Event monitoring support: Input node

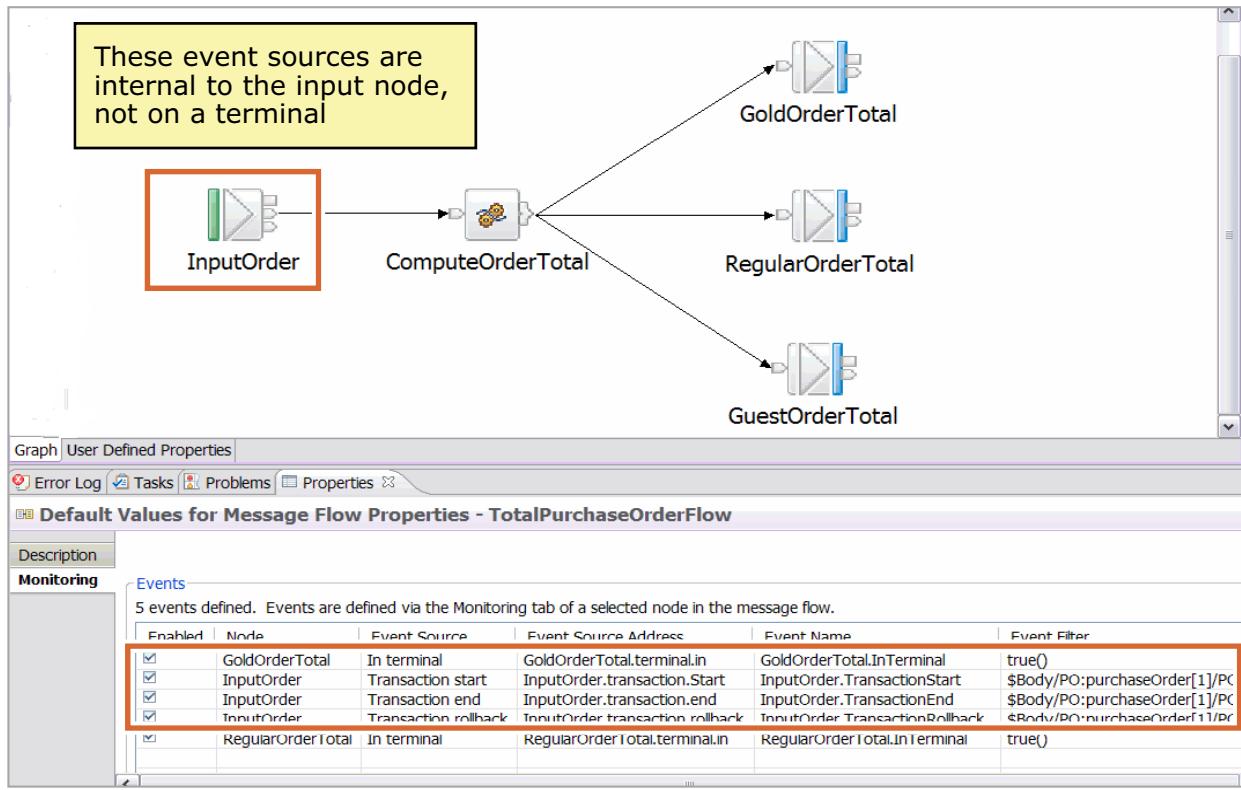


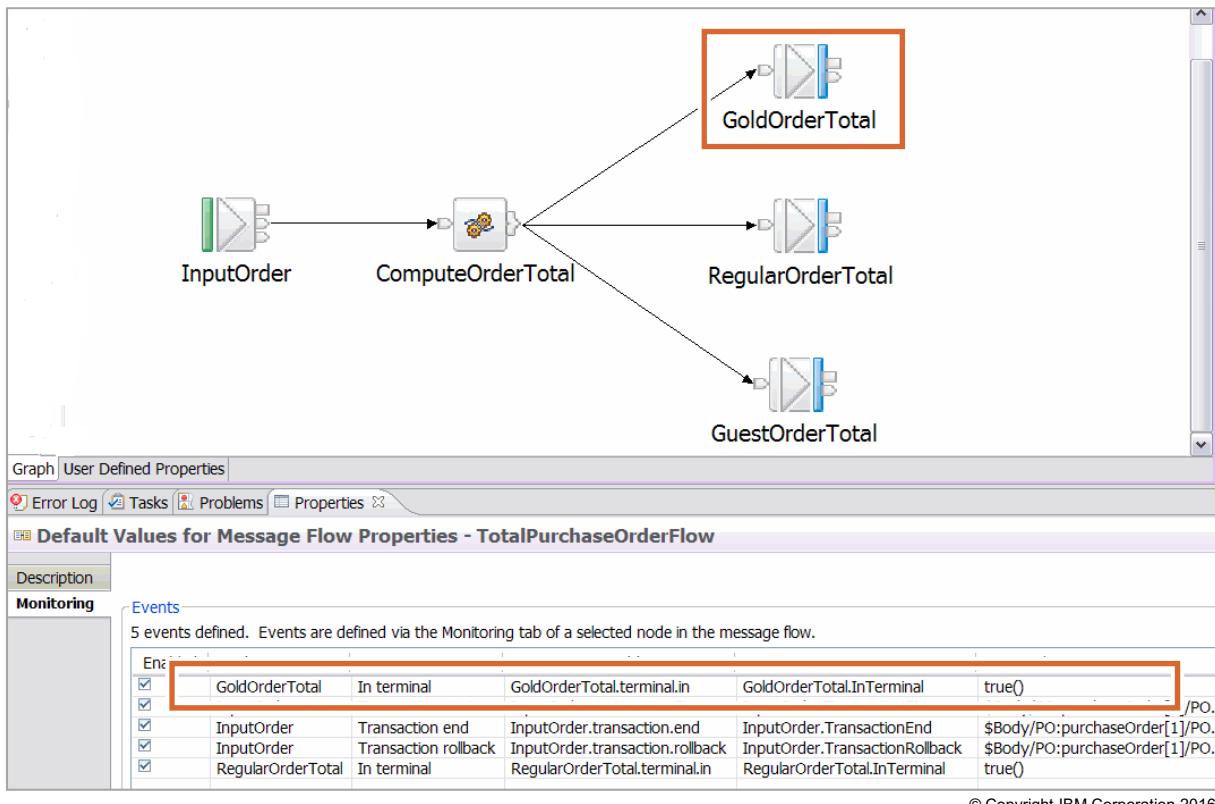
Figure 16-8. Event monitoring support: Input node

WM676/ZM6761.0

### Notes:

This figure highlights the event definitions for the MQ Input node that is named **InputOrder**: transaction start, transaction end, and transaction rollback.

## Event monitoring support: Output node



© Copyright IBM Corporation 2016

Figure 16-9. Event monitoring support: Output node

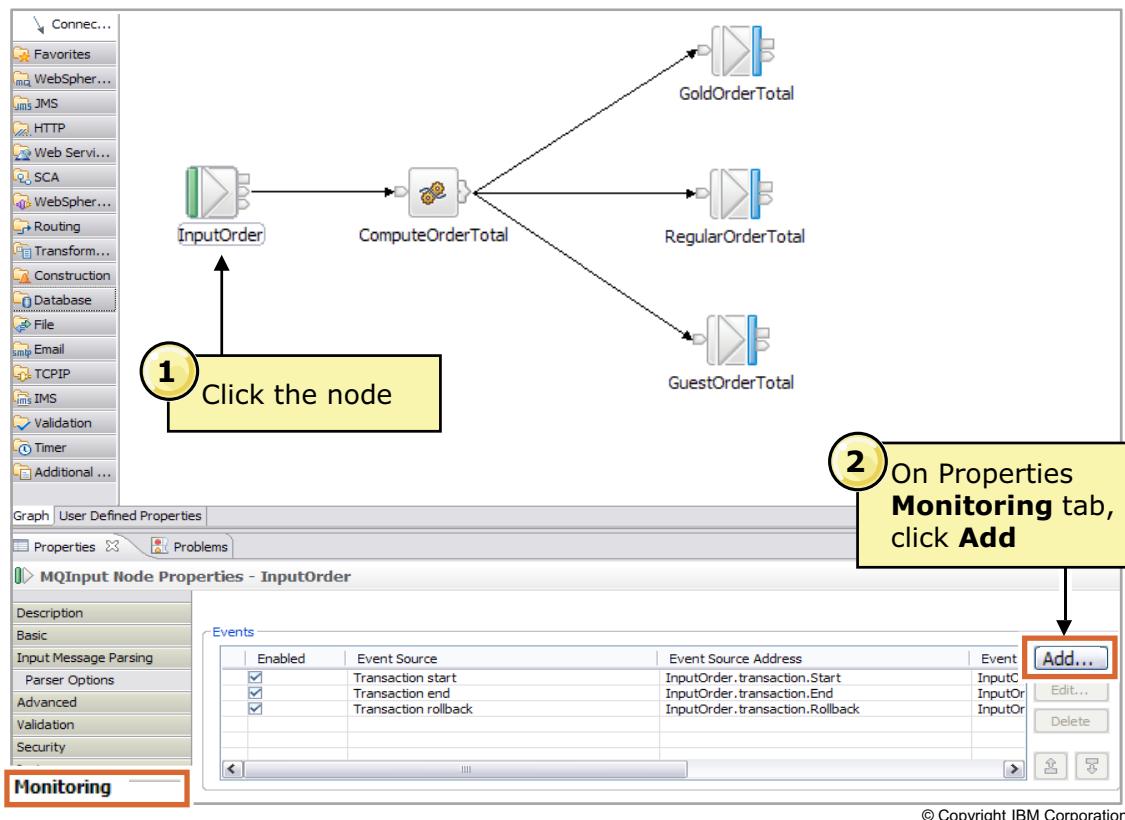
WM676/ZM6761.0

### Notes:

This figure highlights the event that is defined for the MQ Output node that is named **GoldOrderTotal**. The only event that is defined in the node is the **In** terminal, which means that an event is emitted when the message reaches the **In** terminal of the node.



## Adding a monitoring event to a node (1 of 3)



© Copyright IBM Corporation 2016

Figure 16-10. Adding a monitoring event to a node (1 of 3)

WM676/ZM6761.0

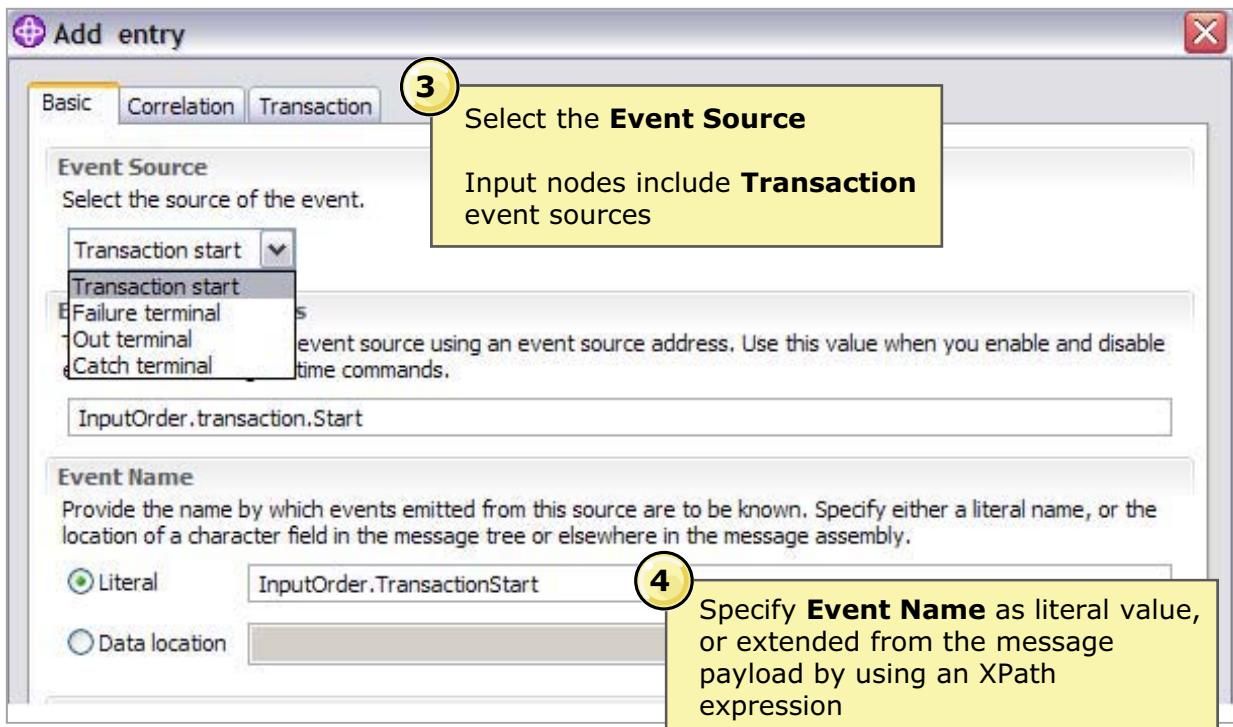
### Notes:

Event configuration is supported in the IBM Integration Toolkit. Events for message flows and nodes are configured on the **Monitoring** tab of the **Properties** view.

- To add, edit, or delete an event for a node, click the node in the flow that generates the events, such as an input node. Click the **Monitoring** tab in the **Properties** view to show the event summary window.
- To add an event, click **Add**.

You can also select any event in the summary and then click **Delete** to delete the event, or click **Edit** to modify the event configuration.

## Adding a monitoring event to a node (2 of 3)



© Copyright IBM Corporation 2016

Figure 16-11. Adding a monitoring event to a node (2 of 3)

WM676/ZM6761.0

### Notes:

- The next step for adding an event is to select the event source from the **Event Source** list. The event source options depend on the node type and terminals.

In the example that is shown here, the node that was selected was an input node with **Failure**, **Out**, and **Catch** terminals. The list contains an event for each output terminal. The list also contains an event for transaction start because the node is an input node.

- Next, specify an **Event Name**. The Integration Toolkit provides an event source literal name, also referred to as the event source address. It is the concatenation of the node name and the event.

In this example, the node name is **InputOrder**. The event that is selected in the previous step was transaction start so the event source literal name is **InputOrder.TransactionStart**. The event source literal name is the name that is used to reference this event from a command.

You have the option of overriding the event name with a literal value or an XPath expression that references an element in the message payload.



## Adding a monitoring event to a node (3 of 3)

**Event Payload**

Most events need to contain data taken from fields in the message tree or from elsewhere in the assembly. Data taken from simple fields or complex fields appears in the event in XML or binary form. Data can also contain bitstream data, which appears in the event as hexadecimal bytes.

**Data location**

\$Root/XMLNSC/PO:purchaseOrder/PO:purchaseOrderID
\$Root/XMLNSC/PO:purchaseOrder/PO:customerID
\$Root/XMLNSC/PO:purchaseOrder/PO:items

**Add...** **Edit...** **Delete**

**5** Click **Add** to add data from headers, payload, or environment

**6** Specify the XPath data location

**Add Data Location**

Data location  Edit...  
✖ Value must not be blank.

OK Cancel

© Copyright IBM Corporation 2016

Figure 16-12. Adding a monitoring event to a node (3 of 3)

WM676/ZM6761.0

### Notes:

The next steps customize the event.

5. The **Event Payload** section identifies information from the payload to provide in the event message. Click **Add**, and then provide the XPath or ESQL path of the payload object in the **Add Data Location** window.
6. If you click **Edit** in **Add Data Location** window, the XPath builder opens with the message structure to assist with creating the XPath for the data.

Multiple XPath queries can be specified by repeating these steps.

Simple fields automatically go into the `applicationData/simpleContent` event type. If a monitoring profile is used, the `@dataType` attribute can be set for each item of `simpleContent`. Even if the Integration Bus tree holds the data as characters, IBM Integration Bus can be instructed to treat it as another data type such as integer or date.

Complex fields automatically go into the `applicationData/complexContent` event type.

Non-XML data from the MRM parser is automatically converted to XML when it is included in a monitoring event.

## Customizing a monitoring event: Bitstream data

**Event Payload**

Most events need to contain data taken from fields in the message tree or from elsewhere in the message assembly. Data taken from simple fields or complex fields appears in the event in XML character format. An event can also contain bitstream data, which appears in the event as hexadecimal bytes.

**Data location**

- \$Root/XMLNSC/PO:purchaseOrder/PO:purchaseOrderID
- \$Root/XMLNSC/PO:purchaseOrder/PO:customerID
- \$Root/XMLNSC/PO:purchaseOrder/PO:items

**Include bitstream data in payload**

**Content** All  
Headers  
Body  
All

**Encoding** base64Binary

**Click to include bitstream data in the payload**

**Specify the bitstream content**

© Copyright IBM Corporation 2016

Figure 16-13. Customizing a monitoring event: Bitstream data

WM676/ZM6761.0

### Notes:

Complete these steps to include the bitstream in the event message:

1. Click the **Include bitstream data in payload** check box in the **Event Payload** section.
2. After the check box is selected, you can select the bitstream content as headers only, body only, or all. You can also select the encoding as Cdata, base64 binary, or hex64 binary.



#### Note

Do not select **CData** for bitstream encoding when **Content** is set to **Headers** or **All** because headers can contain binary data. If you select **CData** for **Encoding**, set **Content** to **Body**.

## Event filter

**Event Filter** is an expression to control whether the event is emitted

Event filters are in the **Summary** table

© Copyright IBM Corporation 2016

Enabled	Node	Event Source	Event Source Address	Event Name
<input checked="" type="checkbox"/>	GoldOrderTotal	In terminal	GoldOrderTotal.terminal.in	GoldOrderTotal.InTermi
<input checked="" type="checkbox"/>	InputOrder	Transaction start	InputOrder.transaction.Start	InputOrder.Transaction
<input checked="" type="checkbox"/>	InputOrder	Transaction end	InputOrder.transaction.End	InputOrder.Transaction
<input checked="" type="checkbox"/>	InputOrder	Transaction rollback	InputOrder.transaction.Rollback	InputOrder.Transaction
<input checked="" type="checkbox"/>	RegularOrderTotal	In terminal	RegularOrderTotal.terminal.in	RegularOrderTotal.InTe

Figure 16-14. Event filter

WM676/ZM6761.0

### Notes:

IBM Integration Bus event filters are used to filter out events that do not match a business rule. So Integration Bus can filter events rather than emitting them to IBM Business Monitor, for example, and filtering them there. This process improves performance by eliminating an unnecessary event.

The event filter can be set to a number, Boolean, or string XPath expression that evaluates to a Boolean true or false. If the expression evaluates to true, then events are emitted. If the expression evaluates to false, then events are not emitted. The default setting is true( ).

As shown in the figure, event filters are configured in the **Event Filters** section that is displayed when you add or edit events in the Integration Toolkit. When you define a filter, you can click **Edit** in the **Event Filter** section to start the XPath expression builder.

The event filters for each event are shown on the **Monitoring** tab **Summary** table. In the example, the event filter for the node **GoldOrderTotal** **In terminal** event source is set to true( ). This value is the default, which causes an event to always be emitted.

## Use an XPath expression to filter event emission

- Expression evaluates to
  - True, event emitted
  - False, event not emitted
- Evaluated at run time
- Set on event source definition
- Expression can reference fields from anywhere in the message assembly
- XPath expression builder support available

© Copyright IBM Corporation 2016

Figure 16-15. Use an XPath expression to filter event emission

WM676/ZM6761.0

### Notes:

With Integration Bus filters, you can specify an XPath or ESQL query that is evaluated at run time to control whether the event should be produced.

For example, if the message flow processes purchase orders and you want an event only if the order was greater than \$10000, configure the event to evaluate the XPath of the purchase order total.

Filters can be configured in the Integration Toolkit or in the exported XML file that contains event information.

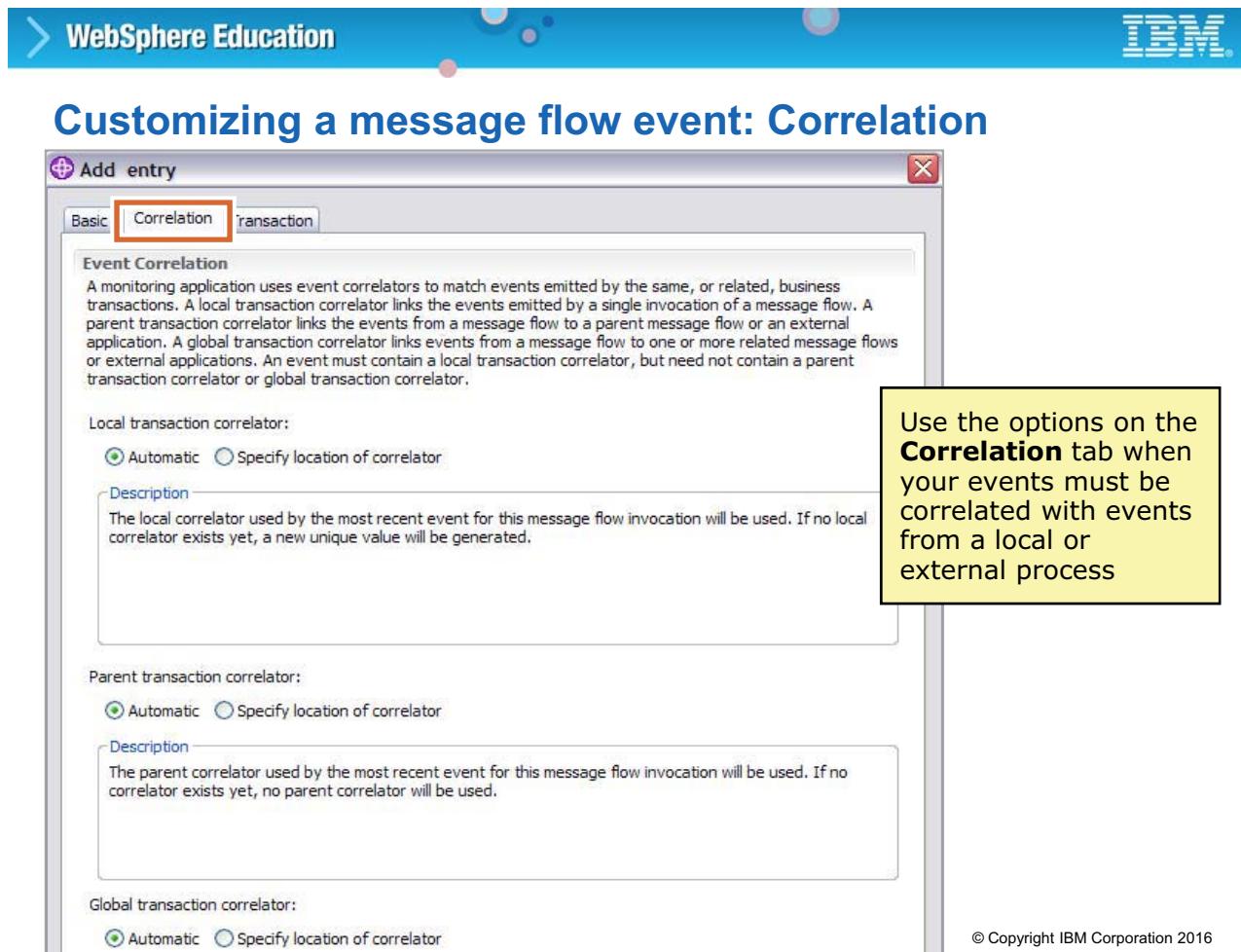


Figure 16-16. Customizing a message flow event: Correlation

WM676/ZM6761.0

## Notes:

Event configuration includes a **Correlation** tab. This option is used to associate events.

Correlation provides three levels of transaction correlator: local, parent, and global.

## Transaction correlators

- A monitoring application uses event correlators to match events that are emitted by the same, or related, business transactions
- Integration Bus developer specifies location in the logical message tree for each correlator on the **Correlation** tab when defining an event in the Integration Toolkit
  - Local transaction correlator links the events that a single invocation of a message flow emits  
Example: Customer ID
  - Parent transaction correlator links the events from a message flow to a parent message flow or an external application  
Example: Order ID
  - Global transaction correlator links events from a message flow to one or more related message flows or external applications  
Example: Stock Amount
- An event must contain a local transaction correlator, but need not contain a parent transaction correlator or global transaction correlator

© Copyright IBM Corporation 2016

Figure 16-17. Transaction correlators

WM676/ZM6761.0

### Notes:

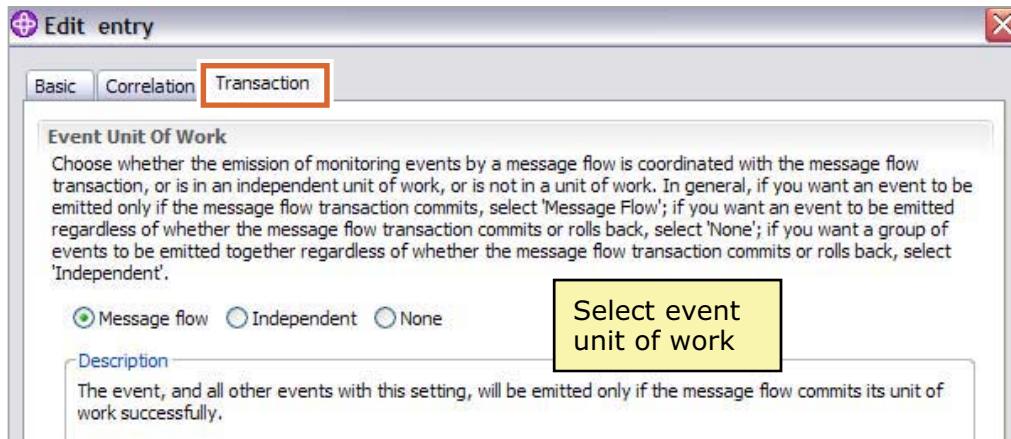
**Local transaction correlator** groups all monitoring events that are related to a specific message. You can select **Automatic** to automatically generate a correlation ID, or you can specify an ID. If you select **Automatic**, Integration Bus reuses the query results from the first evaluation of the XPath query.

**Parent transaction correlator** and **Global transaction correlator** are business-related and correlate events between different messages. For example, if you are processing invoice line items as a transaction, you can correlate to get the invoice number in the parent, then get the individual line item events. Using the same example, you can use **global** to identify the purchase order number and pick up the invoice events from all invoices that reference the purchase order.

 WebSphere Education



## Unit of work



- Use the options on the **Transaction** tab to control the emission of monitoring events by a message flow by selecting one of the following options:
  - **Message flow:** Events are published based on success message flow
  - **Independent:** Groups of events are published together regardless of success or failure of message flow
  - **None:** Events are always published

© Copyright IBM Corporation 2016

Figure 16-18. Unit of work

WM676/ZM6761.0

### Notes:

In the Integration Toolkit, you can choose how event emission is coordinated by specifying a unit of work on the **Transaction** tab, as shown in the figure.

When a message is processed, the IBM MQ updates are included in a unit of work that is referred to as the **Message flow** unit of work. If the message processing is successful, it is committed, and it is rolled back if it fails.

The **Independent** unit of work is a separate unit of work that is created and committed regardless of whether the message is processed successfully or not. Use this option for events, such as those related to error paths, that must be published even if the flow fails.

If you do not want a monitoring event to be included in any unit of work, choose the **None** option.

Some differences in behavior depend on the event type. For example, consider the following events that are generated from a message flow:

- Sequence number 1 is a transaction start event.
- Sequence number 2 is an event in the message flow unit of work.
- Sequence number 3 is an event in the independent unit of work.

- Sequence number 4 is an event that is specified as not in a unit of work.
- Sequence number 5 is an event in the message flow unit of work.
- Sequence number 6 is a transaction end or rollback event.

If the message is successful, all of these events, plus sequence number 6, which is a transaction end event, are published.

If the message fails, events 2 and 5 are rolled back because they belong to the message flow unit of work. Events 1 (transaction start), 3 (independent unit of work), 4 (not in a unit of work), and 6 (a transaction rollback event) are published.

Event 4 (not in the unit of work) is published as soon as it is generated, outside of a unit of work. It is the first to appear externally to Integration Bus and is unaffected by any commit or rollback processing.

## Message flow event monitoring profiles

- Configurable service that is used to customize events after a message flow was deployed, but without redeploying the flow
  - Name: DefaultMonitoringProfile
  - Property: profileProperties Name of the monitoring profile
- An XML document specifies the event sources in a message flow that emits events, and the properties of those events
- Must conform to XML schema file MonitoringProfile.xsd

© Copyright IBM Corporation 2016

Figure 16-19. Message flow event monitoring profiles

WM676/ZM6761.0

### Notes:

You can use a monitoring profile configurable service to customize events after a message flow is deployed without redeploying the flow.



#### Note

If the deployed message flow has event monitoring properties that are configured by using the Integration Toolkit, use the `mqsireportflowmonitoring` command to create the equivalent monitoring profile XML file for the message flow. Use this profile as a starting point for creating other monitoring profiles.

Example: Request a report of the monitoring options for message flow “MyFlow1” in the integration server “default” for integration node “NodeA”:

```
mqsireportflowmonitoring NodeA -e default -f MyFlow1
```

## Example: Message flow event monitoring profile

Example: Include two simple fields from a message

```
<p:monitoringProfile p:version="2.0">
  <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
    <p:applicationDataQuery>
      <p:simpleContent p:dataType="integer" p:name="InvoiceNumber">
        <p:valueQuery p:queryText="$Body/invoice/invoiceNo"/>
      </p:simpleContent>
      <p:simpleContent p:dataType="string" p:name="BatchID">
        <p:valueQuery p:queryText="$Body/batch/batchNo"/>
      </p:simpleContent>
    </p:applicationDataQuery>
  </p:eventSource>
</p:monitoringProfile>
```

© Copyright IBM Corporation 2016

Figure 16-20. Example: Message flow event monitoring profile

WM676/ZM6761.0

### Notes:

This figure shows an example of an event monitoring profile XML file. The profile must conform to the **MonitoringProfile.xsd** schema.

The profile in the example captures the invoice number and batch ID from the body of the logical message. The message is captured at the MQ Input node **Out** terminal (the event source).

## Apply a message flow monitoring profile to a BAR file

Use the Integration Toolkit BAR File editor to apply a monitoring profile to one or more message flows:

1. Double-click the BAR file to open the BAR File editor.
2. Click the **Manage** tab.
3. Select the message flow to display the **Properties** view.
4. In the **Monitoring Profile Name** field, enter the name of a monitoring profile.
5. Save the BAR file.
6. Deploy the BAR file.
7. Activate monitoring for the flow with `mqsicchangeflowmonitoring`.

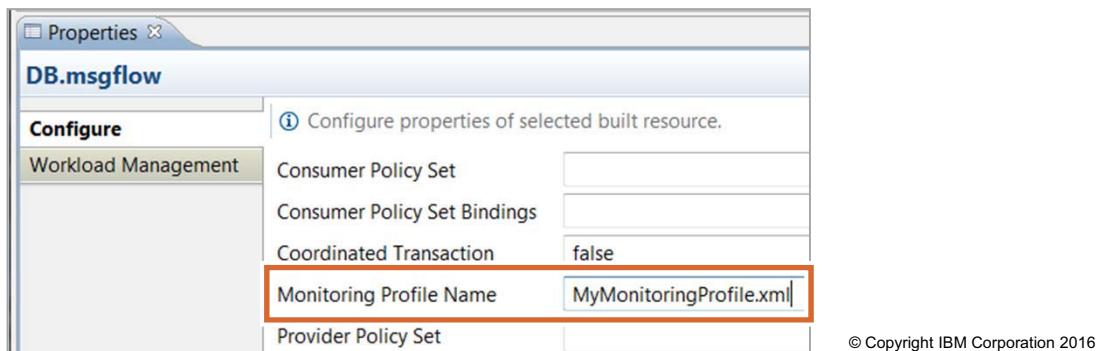


Figure 16-21. Apply a message flow monitoring profile to a BAR file

WM676/ZM6761.0

### Notes:

You can apply a monitoring profile by using a command or by modifying the BAR file.

The figure lists the steps for adding a monitoring profile to a BAR file by using the BAR File editor in the Integration Toolkit.

## 16.2. Record and replay

## Record and replay

- With Integration Bus, you can record messages to a database when they pass through a message flow
- Use for problem determination, auditing, or collection data from a production system for replay on a development system
- To use recorded messages, you must:
  - Configure monitoring on the message flow
  - Create a database to hold the recorded messages
  - Create a data source, and then use a configurable service to define the data source name to use when recording messages
- To replay messages, you must use an existing application to view the messages, or create your own
- Requires that you specify a local IBM MQ queue manager on the integration node

© Copyright IBM Corporation 2016

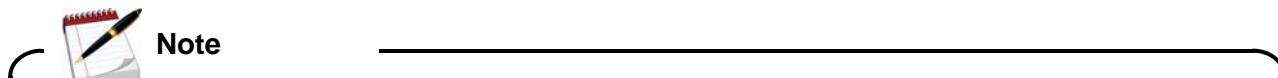
Figure 16-22. Record and replay

WM676/ZM6761.0

### Notes:

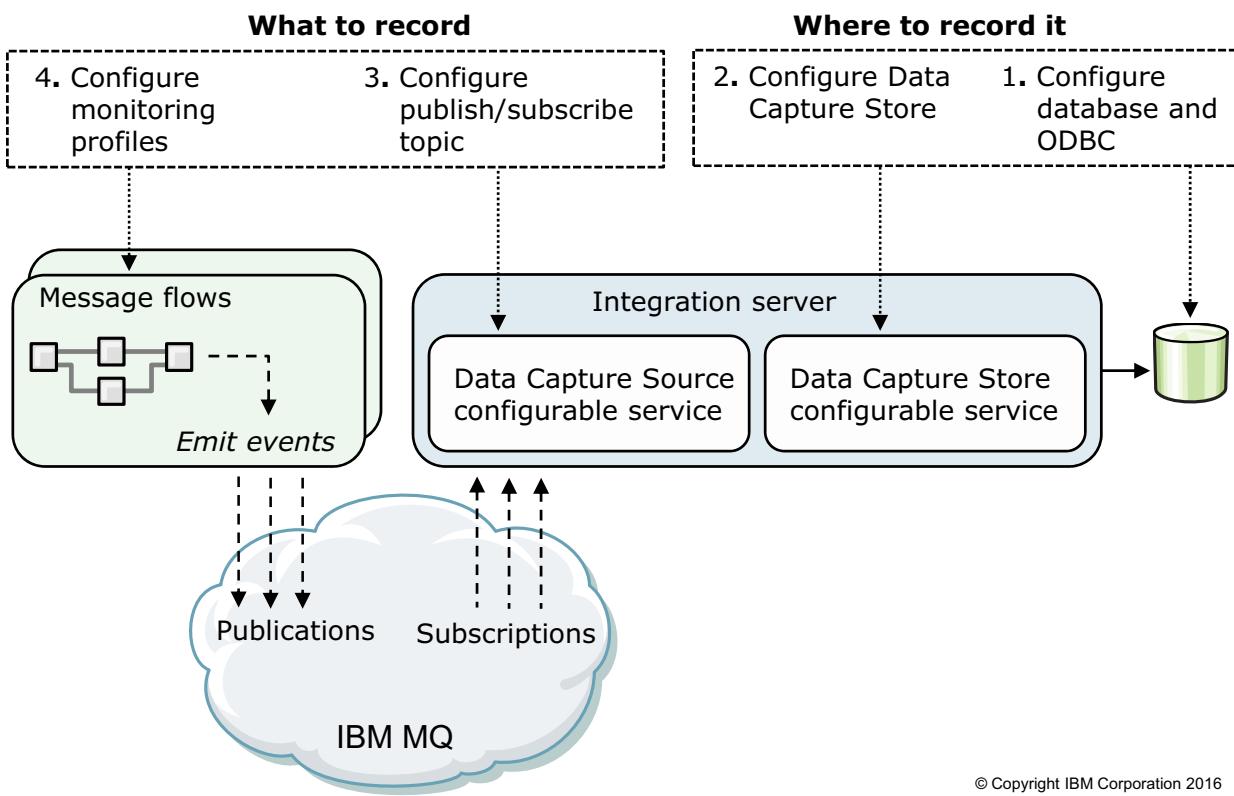
If you configured your message flow to emit event messages, the monitoring events publish selected message data. With the “record and replay” feature of Integration Bus, you can view or replay (resubmit for processing) the message data.

Integration Bus events are typically published to an IBM MQ queue. The record function subscribes to the published monitoring data, and stores the data in a database. You can then view the data through the IBM Integration web interface, or replay it by resending the message to an IBM MQ queue.



The administration tasks for setting up record and replay are covered in detail in course WM646, *IBM Integration Bus V10 System Administration*.

## Record and replay configuration



© Copyright IBM Corporation 2016

Figure 16-23. Record and replay configuration

WM676/ZM6761.0

### Notes:

This figure summarizes the configuration for record and replay.

To configure Integration Bus to record data, complete the following steps. The sequence of these steps is important. If they are not completed exactly as shown, BIP Message BIP2194 is generated on start.

1. Create and configure the database, and define an ODBC definition for the data source name (DSN).
2. To define how and where data is stored, create a DataCaptureStore configurable service. This configurable service specifies the Integration Bus runtime properties for data processing and for connecting to the database.
3. Specify a publish/subscribe topic that identifies the source of the data that you want to capture. To identify the source of the data, create a DataCaptureSource configurable service. You use this configurable service to specify the monitoring topic that identifies the messages flows from which your data comes. This configurable service also identifies the data capture store to use for storing this data.
4. To generate the data that you want to record, configure monitoring on your message flows.

## Preparing to record messages

1. Create a database to hold recorded messages.
2. Create a *DataCaptureStore* configurable service to specify processing information and connection information for the database.
3. Create a *DataCaptureSource* configurable service to identify the monitoring topic and the data capture store to use for processing event data for this topic.
4. Configure monitoring on a message flow to enable events to be emitted for capture, by using monitoring properties or a monitoring profile.

© Copyright IBM Corporation 2016

Figure 16-24. Preparing to record messages

WM676/ZM6761.0

### Notes:

The figure lists the steps that must be completed before you record messages.

First, you must create a database to hold the recorded messages. You must also create an ODBC connection for the database.

You can restrict the users who can view and replay data for an integration node by enabling security. If you do not enable security, all users can complete all actions against an integration node and all integration servers.

The *DataCaptureStore* configurable service identifies the database that holds the messages. The *DataCaptureSource* configurable service identifies the monitoring topics and data capture store. Multiple instances of the *DataCaptureSource* configurable service can use the same *DataCaptureStore* configurable service.

## Creating the database for recorded messages

- Can record data to DB2, Microsoft SQL Server, and Oracle databases
  - IBM Integration Bus includes scripts that create and configure a database that is named MBRECORD for recorded data
- Create an ODBC definition for the database
- Use the `mqsisetdbparms` command to set a user identifier and password for the integration node to use when it connects to the database

© Copyright IBM Corporation 2016

Figure 16-25. Creating the database for recorded messages

WM676/ZM6761.0

### Notes:

You can record data to DB2, Microsoft SQL Server, and Oracle databases.

A script is provided with Integration Bus that you can use to create the database and database tables. You can run this script unmodified, or you can customize it.

The script creates a database that is called MBRECORD with a default schema.



## Viewing the recorded data

- Use the IBM Integration web user interface
- Create an application by using the IBM Integration API
- Create an application by using an IBM Integration Bus REST application programming interface

© Copyright IBM Corporation 2016

Figure 16-26. Viewing the recorded data

WM676/ZM6761.0

### Notes:

You can use the IBM Integration web interface to review the recorded data, update and requeue the message, or delete the message.

You can also use the IBM Integration API or the IBM Integration REST API to write your own message viewer application to fit your requirements.



## IBM Integration web interface: Data viewer tab

The screenshot shows the IBM Integration web interface's Data viewer tab. The table displays monitoring events with the following data:

	Event time	Local Transaction ID	Parent Transaction ID	Global Transaction ID	Data	Errors	Event name
<input type="checkbox"/>	2013-05-31 13:41:35.972	CG123456	BNY347290	IBM \$1000		-	Gold customer: Processing trade
<input type="checkbox"/>	2013-05-31 13:41:35.834	CG123456	BNY347290	IBM \$1000		-	Deciding customer type
<input type="checkbox"/>	2013-05-31 13:41:35.967	CG123456	BNY347290	IBM \$1000		-	Decision: Gold customer
<input type="checkbox"/>	2013-05-31 13:41:44.661	CR100200-A	BNY809092	APPL \$500		-	Deciding customer type
<input type="checkbox"/>	2013-05-31 13:41:50.250	GU123456	BNY348475	MSFT \$5000		-	Deciding customer type
<input type="checkbox"/>	2013-05-31 13:41:50.251	GU123456	BNY348475	MSFT \$5000		-	Decision: Guest customer
<input type="checkbox"/>	2013-05-31 13:41:35.773	CG123456	BNY347290			-	Trade instruction received
<input type="checkbox"/>	2013-05-31 13:41:44.659	CR100200-A	BNY809092			-	Trade instruction received
<input type="checkbox"/>	2013-05-31 13:41:50.250	GU123456	BNY348475			-	Trade instruction received

© Copyright IBM Corporation 2016

Figure 16-27. IBM Integration web interface: Data viewer tab

WM676/ZM6761.0

### Notes:

The **Data viewer** tab in the IBM Integration web interface contains the monitoring events. You can customize the headings so that they are more descriptive.

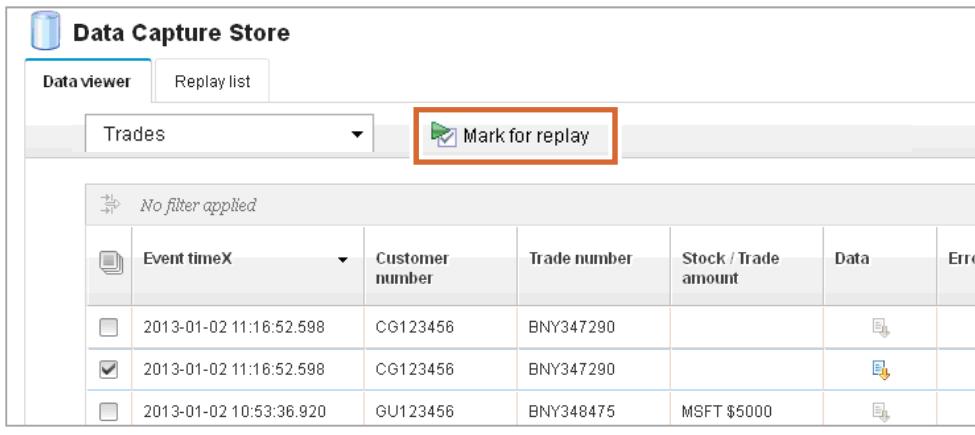
- You can change the name of each column by double-clicking the column name, and then entering another name. These changes are stored in the Integration Registry and are retained uniquely for each data capture store. All users who view data from the same data capture store see the changes that this user made. If you want to record and view data with different headings, record the events in a separate data capture store.
- You can select or clear any of the recorded fields that are shown in the table.
- You can override the width of the column by using the divider bars.

You can also limit the data that is displayed in the **Data viewer** by using the **Filter** function.

## Replaying messages

1. Select the Data Capture Store that contains the data to replay.
2. Select the messages to replay by clicking the check box next to each message.
3. Click **Mark for replay** to add messages to the replay list.
4. Click the **Replay list** tab.
5. Select the destination for the messages.
6. To replay all messages, click **Replay all**.  
To replay a single message, click the **Replay** icon in the row.



The screenshot shows the 'Data Capture Store' interface. At the top, there are tabs for 'Data viewer' and 'Replay list', with 'Data viewer' currently selected. Below the tabs is a dropdown menu set to 'Trades'. To the right of the dropdown is a button labeled 'Mark for replay' with a checkmark icon, which is highlighted with a red box. Underneath these controls is a table with the following columns: Event timeX, Customer number, Trade number, Stock / Trade amount, Data, and Error. There are three rows of data:

	Event timeX	Customer number	Trade number	Stock / Trade amount	Data	Error
<input type="checkbox"/>	2013-01-02 11:16:52.598	CG123456	BNY347290			
<input checked="" type="checkbox"/>	2013-01-02 11:16:52.598	CG123456	BNY347290			
<input type="checkbox"/>	2013-01-02 10:53:36.920	GU123456	BNY348475	MSFT \$5000		

© Copyright IBM Corporation 2016

Figure 16-28. Replying messages

WM676/ZM6761.0

### Notes:

The figure lists the steps for replaying messages by using the IBM Integration web console.

Each row that is displayed represents a recorded message. To sort these rows into a particular order, click a column heading. You can also use a filter to help find rows in which you are interested.

Filtering is case-sensitive and allows the use of wildcards. To search for an exact match, enclose the search string in quotation marks. You can also search for a substring, and you can use an asterisk (\*) to match zero or more characters.

## Unit summary

Having completed this unit, you should be able to:

- Define monitoring events in the message flow
- Use the record and replay function to capture and review processed messages

© Copyright IBM Corporation 2016

Figure 16-29. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: Using monitoring profiles for recording events is generally considered to be more flexible than setting monitoring properties.
  
2. True or False: The record-and-replay function in Integration Bus requires that an IBM MQ server is installed on the same computer as the integration node, and that you specify a queue manager on the integration node.

© Copyright IBM Corporation 2016

Figure 16-30. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
  
- 2.

## Checkpoint answers

1. True or false: Using monitoring profiles for recording events is generally considered to be more flexible than setting monitoring properties.

Answer: **True**

2. True or False: The record-and-replay function in Integration Bus requires that an IBM MQ server is installed on the same computer as the integration node, and that you specify a queue manager on the integration node.

Answer: **True**

© Copyright IBM Corporation 2016

Figure 16-31. Checkpoint answers

WM676/ZM6761.0

### Notes:

## Exercise 8



Recording and replaying message  
flow data

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

10.1

Figure 16-32. Exercise 8

WM676/ZM6761.0

### Notes:

In this exercise, you define message flow events and then record them in a database. You also replay messages and capture and report failed events.



## Exercise objectives

After completing this exercise, you should be able to:

- Define monitoring events
- Activate flow monitoring
- View event messages in the IBM Integration web user interface
- Replay messages
- Capture and report failed events

© Copyright IBM Corporation 2016

Figure 16-33. Exercise objectives

WM676/ZM6761.0

### Notes:

See the Student Exercises guide for detailed instructions.



# Unit 17. Managing the workload

## What this unit is about

Workload management allows system administrators to monitor and adjust the speed with which messages are processed, and control the actions that are taken on unresponsive flows and threads. A workload management policy defines properties on the message flow in the Integration Registry so that message flows can refer to the policy to find properties at run time. This unit describes workload management and how to use workload management policies to specify goals for the processing rate of a message flow.

## What you should be able to do

After completing this unit, you should be able to:

- Describe the workload management options for adjusting the speed with which messages are processed, and controlling the actions that are taken on unresponsive flows and threads
- Use a workload management policy to control the workload management attributes at run time

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center for IBM Integration Bus V10



## Unit objectives

After completing this unit, you should be able to:

- Describe the workload management options for adjusting the speed with which messages are processed, and controlling the actions that are taken on unresponsive flows and threads
- Use a workload management policy to control the workload management attributes at run time

© Copyright IBM Corporation 2016

---

Figure 17-1. Unit objectives

WM676/ZM6761.0

### Notes:

## Control the processing speed in Integration Bus

- Message flow processing rate control provides intelligent mechanisms to increase and decrease processing speed
- Policy specifies goals for the processing rate of a message flow
- Allow more diverse workload management:
  - Apply policy to different Integration Bus artifacts such as applications, services, and individual nodes
  - Schedule policy application
- Allow workload management to be defined in a policy separate from the message flow
  - Define policy in the BAR file, on the message flow, or in the Integration Registry
  - Use common repository to store policies in the integration node or somewhere else
  - Policy can be changed and used in integration node independently of where it is stored
- Requires that event publication is enabled and that a publish/subscriber broker is configured

© Copyright IBM Corporation 2016

Figure 17-2. Control the processing speed in Integration Bus

WM676/ZM6761.0

### Notes:

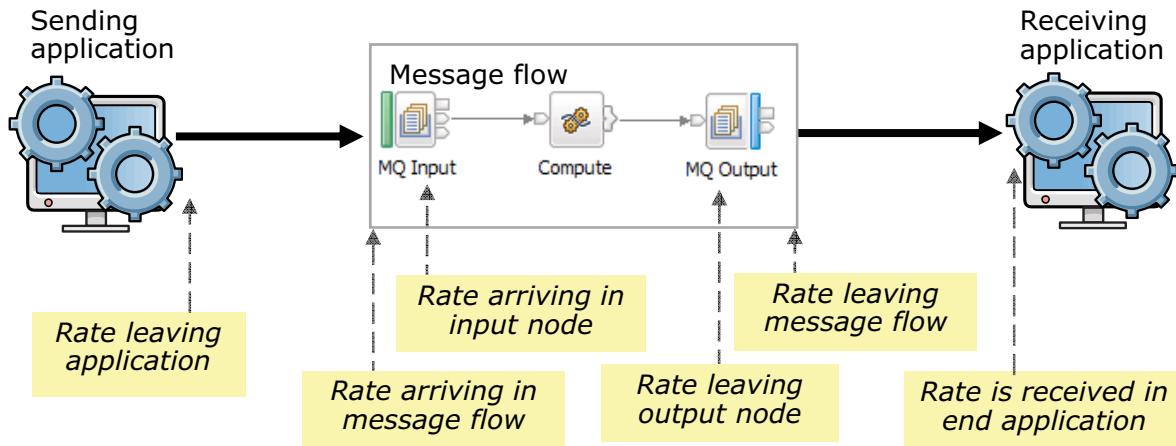
With Integration Bus, system administrators can monitor and adjust the speed that messages are processed, and control the actions that are taken on unresponsive flows and threads.

With workload management, the system administrator can specify a notification threshold for message flows. If the notification threshold is exceeded, an out of range notification message is produced. If the notification threshold later drops back into range, a back in range notification message is produced.

The system administrator can set the maximum rate at which an individual message flow can run. The maximum rate is specified as the total number of input messages processed every second. When set, the number of input messages that are processed across the flow is measured.

The system administrator can specify a workload management policy for a message flow. The workload management policy encompasses all of the properties available under workload management in one place and allows for easier tuning of message flow performance.

## Message flow processing rate control terms



- At various points in the flow of data from one application to another, the message rate can be measured and controlled
- Rate arriving in message flow** is used as the controlling rate for the message flow
  - Effectively limits **Rate received in end application**
  - Includes the total rates of all input nodes of any type in the message flow

© Copyright IBM Corporation 2016

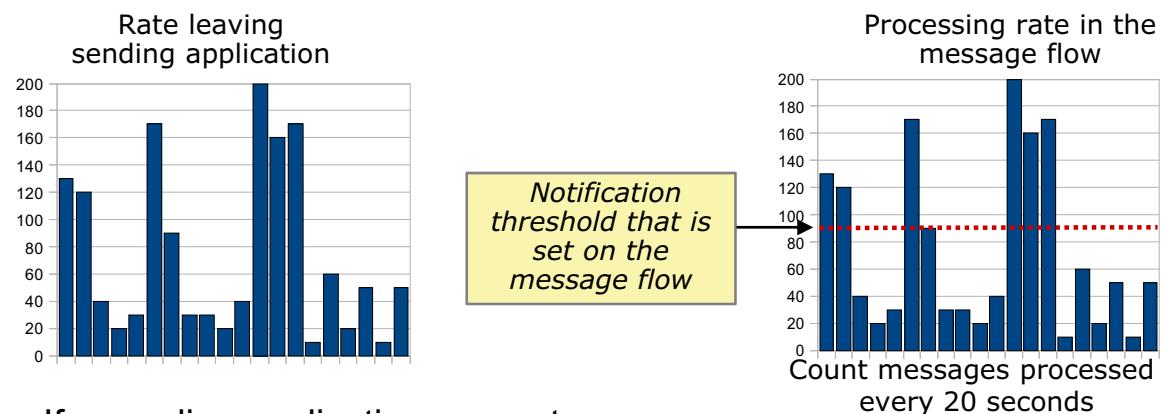
Figure 17-3. Message flow processing rate control terms

WM676/ZM6761.0

### Notes:

The figure shows the various points in the flow of data from one application to another that the message rate can be measured and controlled. The rate at which the messages arrive into the message flow is used as the controlling rate for the message flow.

## Message flow processing rate control: Notification



- If a sending application generates a higher (or lower) than expected message rate, configure the integration node to send a notification when a threshold is exceeded or it drops below the threshold
  - Publish/subscribe mechanism decouples notification from consumers
  - Write to Activity log
  - Write to User trace
- Action to reduce message rate is up to the user:
  - Slow the sending application
  - Email administrator to investigate
  - Stop message flow to protect receiving application

© Copyright IBM Corporation 2016

Figure 17-4. Message flow processing rate control: Notification

WM676/ZM6761.0

### Notes:

With workload management, the system administrator can specify a notification threshold for message flows. The notification threshold is a measure of the total messages every second.

Before activating the message flow notification threshold, the system administrator must first establish the message rate that is required for the message flow. The message rate is a measure of the total number of messages that arrive each second at the message flow from all input nodes of any type. It is the total rate and not the average rate.

Consider an example of how the message flow is determined: A message flow contains two input nodes that are called A and B. Assume that input node A received messages at the rate of 50 messages a second, and input node B received messages at the rate of 70 messages a second. The total message rate for the message flow would be 120 messages per second.

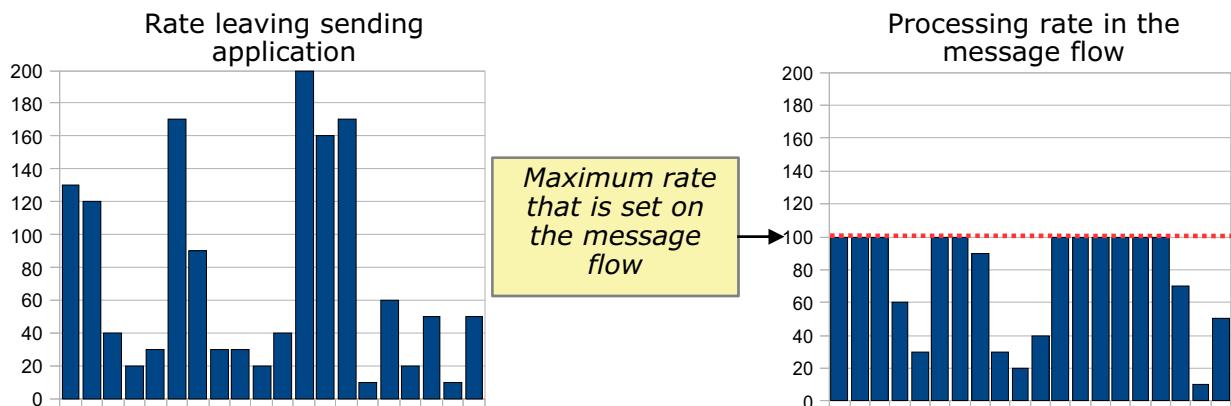
A notification threshold value of zero, or not set, causes the message flow notification threshold to be turned off. The default state is off.

The notification threshold can be set for a message flow in two ways:

- Directly within a BAR file by setting a BAR file property

- As one of the attributes within a workload management policy that is defined within the Integration Registry

## Message flow processing rate control: Delay



- A sending application generates a “bursty” message rate
- A receiving application that uses the messages can handle the average rate but not short-term fluctuations
- Configure message flow to:
  - Limit the rate on a message-by-message basis
  - Delay messages if going too fast

- Time is measured before getting the message
- Time is measured again before getting next message
- If time difference is less than the time required to keep to the maximum rate, delay is made

© Copyright IBM Corporation 2016

Figure 17-5. Message flow processing rate control: Delay

WM676/ZM6761.0

### Notes:

The system administrator can set the maximum rate at which an individual message flow can run. The maximum rate is specified as the total number of input messages processed every second. This measure is irrespective of the number of instances in use, the number of input nodes in the message flow, or the number of errors that occur. If necessary, an automatic processing delay keeps the input message processing rate under the maximum flow rate setting.

The maximum rate value is divided equally among all threads that are running in the message flow irrespective of the number of input nodes within the flow.

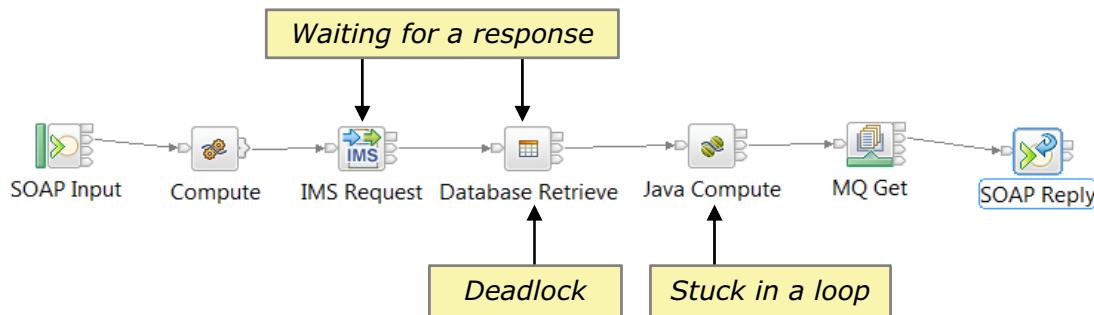
To calculate the number of threads within a specific message flow:

1. Count the number of input nodes within the flow.
2. Add the number of instances that are specified for each input node.

The following example assumes that the maximum rate is set to 50. A message flow has three input nodes with four instances that are specified on the first node and three instances that are specified on the second node. Ten threads are in operation: three input node threads, four instance threads for the first node, and three instance threads for the second node. The maximum rate is split equally across the 10 threads. Each thread has a maximum rate allocation of five messages every second.

You can set the maximum rate for a message flow directly in the BAR file or as one of the attributes within a workload management policy that is defined within the Integration Registry.

## Unresponsive flows



- Message flow processing can become unresponsive when:
  - Waiting for a response from an external system
  - Processing an infinite loop or a calculation that takes a long time
  - Deadlocked between two resources

© Copyright IBM Corporation 2016

Figure 17-6. Unresponsive flows

WM676/ZM6761.0

### Notes:

With workload management, you can also specify and monitor the maximum amount of time that any message flow is allowed to process a message, and specify an action to be taken if the timeout is exceeded.

A message flow can become unresponsive for three primary reasons:

- The message flow is waiting for a response from an external system such as a database, website, or application.
- The message flow is stuck in a loop because of faulty logic or a missing dead-letter queue.
- The message flow is deadlocked between two resources.

## Handling unresponsive flows

- Programmatically check from within a message flow if it was requested to stop
- Manually force a message flow to stop from a command with a forced restart of the integration server (execution group)

Example: `mqsistopmsgflow IBNODE -e is1 -m mf1 -w 30  
-f restartExecutionGroup`

- Automatically force a message flow to stop by setting message flow properties in BAR file or Workload Management policy:
  - Processing Timeout:** Maximum time in seconds that a message flow can process a message before acting
  - Processing Action:** The action to take when the **Processing Timeout** is exceeded. Values are **None** or **Restart execution group** (integration server)

© Copyright IBM Corporation 2016

Figure 17-7. Handling unresponsive flows

WM676/ZM6761.0

### Notes:

There are three ways to handle unresponsive flows.

- Use the ESQL, Java, or .NET programming APIs to programmatically check whether a message flow was requested to stop. You can use a function in the programming API to check from within a message flow when the flow is requested to stop. If a request is made to stop the message flow, the function in all three APIs returns a Boolean value of `true`.
- Manually force a message flow to stop. The force option `-f restartExecutionGroup` on the `mqsistopmsgflow` command flags the message flow as “stopped” and then restarts the integration server (execution group). When the integration server restarts, the message flow is in the “stop” state.
- Use the **Processing Timeout** and **Processing Action** workload management properties on the BAR file or in an applied Workload Management policy to automatically force a message flow to stop. **Processing Timeout** is the maximum time a message flow can process a message before acting. **Processing Action** is the action to take (“none” or “restart execution group”). An event message is published on an IBM MQ topic after the processing timeout is exceeded and again when the message flow processing finishes.

## Workload Management properties

- **Notification Threshold** (messages per second)
  - Rate at which notification is published when it is exceeded
  - Default: Infinite
- **Maximum Rate** (messages per second)
  - Maximum rate a flow processes message
  - Default: Infinite
- **Processing Timeout and Processing Action**

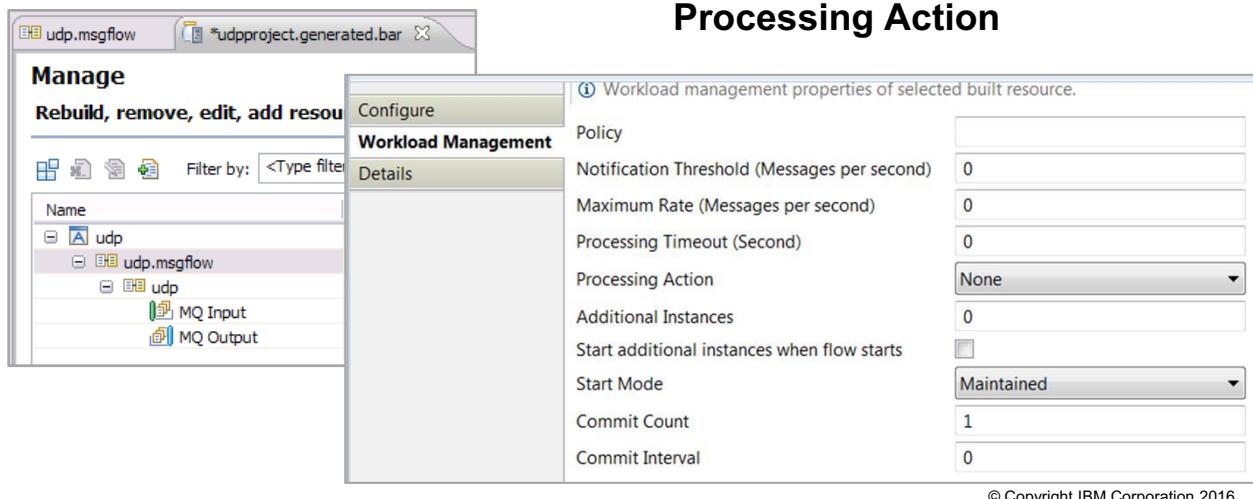


Figure 17-8. Workload Management properties

WM676/ZM6761.0

### Notes:

The **Workload Management** properties can be modified in the BAR file editor in the Integration Toolkit. Click the message flow or a message processing node and then click **Workload Management** on the **Properties** tab.

## Processing Timeout notification topics

- Integration Bus publishes messages for:
  - *Processing timeout* alerts when the message flow processing timeout period is exceeded
  - *Processing finished* alerts when message flow processing is complete and **Processing Action** is set to **None**
- Before you can use workload management functions:
  - Ensure that the publication of events is enabled
  - Ensure a publish/subscribe broker is configured

Examples:

- View the *Processing timeout* message that is published to IBM MQ by subscribing to the topic:

```
$SYS/Broker/<IntNode>/WorkloadManagement/ProcessingTimeout/<IntServer>/  
application/messageFlow
```

- View the *Processing finished* message that is published to IBM MQ by subscribing to the topic:

```
$SYS/Broker/<IntNode>/WorkloadManagement/ProcessingFinished/<IntServer>/  
application/messageFlow
```

© Copyright IBM Corporation 2016

Figure 17-9. Processing Timeout notification topics

WM676/ZM6761.0

### Notes:

Integration Bus publishes messages for various workload conditions. Two of those publications are for **Processing Timeout** and **Processing Finished**:

- **Processing Timeout** alerts when the message flow processing timeout period is exceeded.
- **Processing Finished** alerts when message flow processing is complete and no timeout action is specified (the **Processing Action** property is set to **None**).

The topic strings are available for subscriptions to receive the alerts. You can define subscriptions with IBM MQ Explorer or write your own applications to subscribe to the publications for the integration servers, applications, and message flows that interest you.

For example, you can view the **Processing Timeout** alert message by subscribing to the following topic:

```
$SYS/Broker/<IntNode>/WorkloadManagement/ProcessingTimeout/<IntServer>/  
<application>/<library>/<messageFlow>
```

Where:

- *<IntNode>* is the name of the integration node

- *<IntServer>* is the name of the integration server on that integration node
- *<application>* is the name of the application on that integration server
- *<library>* is the name of the library on that application
- *<messageFlow>* is the name of the message flow that is deployed to the library

When the message flow is not contained in either an application or a library, the *<application>* or *<library>* parameters must be omitted along with their enclosing forward slash (/). For example, if the message flow is contained in an application and not in a library, the syntax is:

```
$SYS/Broker/<IntNode>/WorkloadManagement/ProcessingTimeout/<IntServer>/  
<application>/<messageFlow>
```

If the **Processing Action** property is set to **None** and processing of the message flow continues to completion, another event message is published to indicate that the processing is finished. You can view the message by subscribing to the following topic:

```
$SYS/Broker/<IntNode>/WorkloadManagement/ProcessingFinished/<IntServer>/  
<application>/<library>/<messageFlow>
```

If the **Processing Action** property is set to **Restart the execution group**, the execution group (integration server) is restarted and no further event messages are published from the message flow.

## Processing Timeout alert example

```
<wmb:event xmlns:wmb="http://www.ibm.com/xmlns/prod/websphere/messages</pre>
```

© Copyright IBM Corporation 2016

Figure 17-10. Processing Timeout alert example

WM676/ZM6761.0

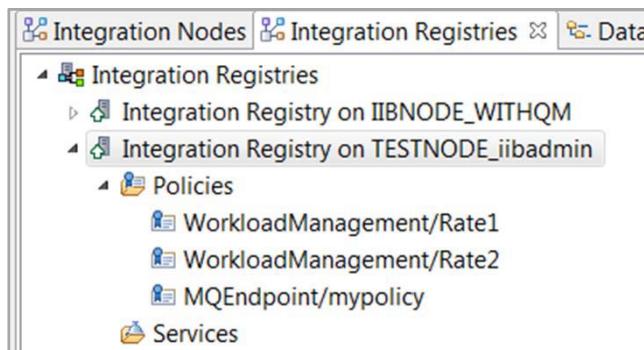
### Notes:

The figure shows an example of the XML message for a **Processing Timeout** event.

The XML message identifies the execution group (integration server) and message flow. The message also contains the processing timeout value for the message flow and the processing timeout action. It also reports the time that is taken, the thread ID, and the node trail.

## Workload Management policies in the Integration Registry

- Registry local to integration node
- Allows the registry to become part of the integration solution
- Use IBM Integration web interface to manage policies and services
- Use IBM Integration Toolkit to view
- Based on standards such as Open Services for Lifecycle Collaboration (OSLC) and linked data to make it flexible



© Copyright IBM Corporation 2016

Figure 17-11. Workload Management policies in the Integration Registry

WM676/ZM6761.0

### Notes:

Integration Bus provides an Integration Registry, which can store workload management policies and service definitions.

The Integration Registry is hosted inside the integration node. You can control whether an integration node offers an Integration Registry by using the IBM Integration web interface. All new integration nodes that are created in Integration Bus have an Integration Registry available by default.

# WebSphere Education

## Using the Integration web interface to manage policies

**Targets and Limits**

- Notification Threshold: 17 messages per second
- Maximum Rate: 20 messages per second

**Additional Instances**

- Integration web interface can be used to create, update, retrieve, and delete policies
- Policies can be attached and detached from message flows

**Unresponsive Message Flows**

- Processing timeout action: Restart execution group
- Processing timeout: 60 seconds

Figure 17-12. Using the Integration web interface to manage policies

WM676/ZM6761.0

### Notes:

Instead of defining Workload Management properties on the BAR file, you can create policies so that message flows can refer to them at run time. If you use this method, you can change the values of attributes for a policy on the integration node, which then affects the behavior of a message flow without the need for redeployment.

The Workload Management policy encompasses all of the properties available under workload management in one place. The policy includes the notification threshold and maximum rate, which allows for easier tuning of message flow performance.

A policy can be set up and administered within the IBM Integration web interface.

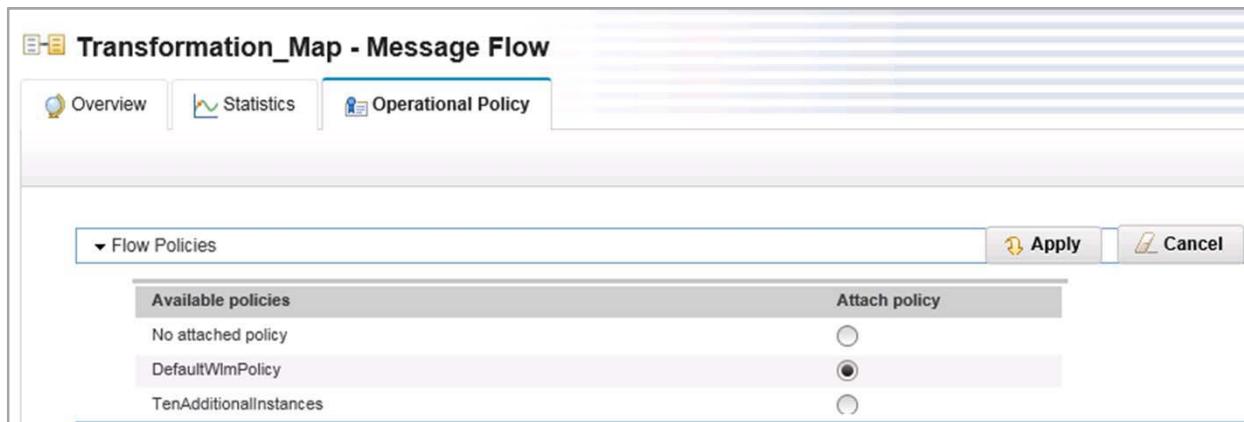
When a policy is created and deployed, it adheres to the following precedence rules:

- Properties that are set in the policy take precedence over properties that are set as BAR file properties.
- If a property is not set in the policy, the equivalent property that is set on the BAR file takes effect.



## Using the Integration web interface to attach a policy

1. In the Integration web interface, click the down arrow on right side of the application or message flow, and then click **Attach Policy**.
2. On the **Operational Policy** tab, click **Attach policy** for the workload management policy.
3. Click **Apply**.



© Copyright IBM Corporation 2016

Figure 17-13. Using the Integration web interface to attach a policy

WM676/ZM6761.0

### Notes:

You can attach a workload management policy to a message flow or application in the IBM Integration web interface.



## Unit summary

Having completed this unit, you should be able to:

- Describe the workload management options for adjusting the speed with which messages are processed, and controlling the actions that are taken on unresponsive flows and threads
- Use a workload management policy to control the workload management attributes at run time

© Copyright IBM Corporation 2016

---

Figure 17-14. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: A Workload Management policy can be created in the IBM Integration Toolkit.
  
2. Complete the sentence: \_\_\_\_\_ controls the rate for the message flow
  - A. Rate leaving application
  - B. Rate arriving in input node
  - C. Rate arriving in message flow

© Copyright IBM Corporation 2016

Figure 17-15. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

1.

2.



## Checkpoint answers

1. True or false: A Workload Management policy can be created in the IBM Integration Toolkit.

Answer: **False.** A Workload Management policy can be viewed in the IBM Integration Registries view in the Integration Toolkit but it cannot be created in the Integration Toolkit.

2. Complete the sentence: \_\_\_\_\_ controls the rate for the message flow
  - A. Rate leaving application
  - B. Rate arriving in input node
  - C. Rate arriving in message flow

Answer: **C**

© Copyright IBM Corporation 2016

---

Figure 17-16. Checkpoint answers

WM676/ZM6761.0

### Notes:

# Unit 18. Creating patterns for reusability

## What this unit is about

This unit describes how to create and test user-defined message flow patterns in the IBM Integration Toolkit.

## What you should be able to do

After completing this unit, you should be able to:

- Construct and extend a user-defined pattern
- Create a pattern authoring project
- Build pattern plug-ins
- Package and distribute pattern plug-ins
- Install a pattern archive

## How you will check your progress

- Checkpoint
- Lab exercises

## References

IBM Knowledge Center for IBM Integration Bus V10

## Unit objectives

After completing this unit, you should be able to:

- Construct and extend a user-defined pattern
- Create a pattern authoring project
- Build pattern plug-ins
- Package and distribute pattern plug-ins
- Install a pattern archive

© Copyright IBM Corporation 2016

---

Figure 18-1. Unit objectives

WM676/ZM6761.0

### Notes:



## Patterns overview

- A **pattern** is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context
  - Faster, more efficient solution development
  - Higher quality, well-architected solutions
- IBM Integration Bus patterns that implement many common application solutions are available on OT4I GitHub Pattern Repository

© Copyright IBM Corporation 2016

Figure 18-2. Patterns overview

WM676/ZM6761.0

### Notes:

A pattern captures a tested solution to a common application problem or specific business problem. A catalog of common Integration Bus application solution patterns is available on the OT4I GitHub Pattern Repository.

The patterns are divided into pattern categories. Pattern categories are categories that are based on the pattern classification and structure that is shown in the **Patterns Explorer**. The catalog provides detailed help that guides you toward a suitable IBM Integration Bus pattern to create resources that are used to solve a specific business problem.

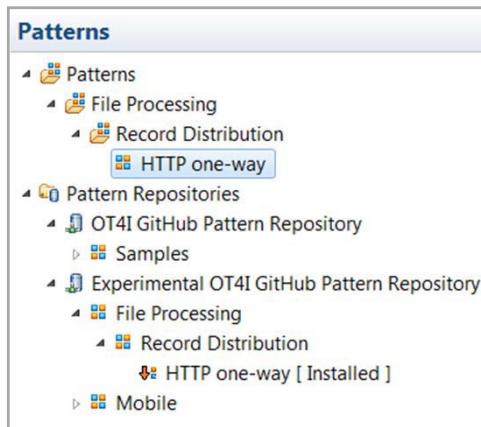
When you start your Integration Bus development by using a pattern:

- Your applications are based on well-designed solutions that are analyzed and optimized
- You can decrease your development time because you are not creating the project “from scratch”
- You can learn preferred techniques for developing application solutions

Integration Bus patterns and the Patterns Explorer were introduced in the prerequisite course WM666, *IBM Integration Bus V10 Application Development I*.

## Using patterns from the GitHub repository

1. Download pattern from OT4I GitHub Pattern Repository.
2. Review the **Pattern Specification** to:
  - Learn what the pattern does
  - See constraints or limitations on using the pattern
  - Review any prerequisite tasks, constraints, and limitations
  - Understand the tasks that must be completed after implementing the pattern
3. Click **Create New Instance** in the Patterns Explorer.
4. Configure the pattern by setting *pattern parameters*.
5. Generate a *pattern instance project*.



© Copyright IBM Corporation 2016

Figure 18-3. Using patterns from the GitHub repository

WM676/ZM6761.0

### Notes:

This figure shows the steps for getting patterns from the GitHub Pattern Repository.

The pattern instance project contains a pattern instance configuration file that stores the configuration parameter values that you used to generate each instance of a pattern. You can review this file and optionally regenerate the pattern instance. Regenerating a pattern instance deletes the IBM Integration Bus projects that were created from the previous generation process and re-creates them. The pattern instance configuration file provides documentation of the parameter settings that were used to generate an instance.

## User-defined patterns overview

- In addition to the various patterns that IBM provides, you can create user-defined patterns
- To use a user-defined pattern, a user downloads the pattern archive and installs it from the IBM Integration Toolkit

© Copyright IBM Corporation 2016

Figure 18-4. User-defined patterns overview

WM676/ZM6761.0

### Notes:

Like many components of IBM Integration Bus, the concept of patterns is extensible. In addition to taking advantage of a number of built-in patterns, you can also create user-defined patterns.

After you create a user-defined pattern, you can package and distribute it. A user then adds this user-defined pattern to the Integration Toolkit, where it can be used like any other pattern. This extensibility can help you to extend good practices within your organization by providing predefined templates from which to start development.

## Process for creating a user-defined pattern

### Integration developer

- Develops exemplar message flow
- Tests pattern

### Pattern user

- Installs pattern
- Browses Pattern Explorer
- Enters pattern parameters
- Generates pattern instance
- Customizes pattern instance
- Deploys pattern instance
- Tests pattern instance

### Pattern author

- Defines target properties
- Selects files and resources
- Defines pattern user instance
- Builds pattern plug-ins
- Tests pattern
- Distributes pattern

© Copyright IBM Corporation 2016

Figure 18-5. Process for creating a user-defined pattern

WM676/ZM6761.0

### Notes:

The IBM Integration Bus developer, the pattern author, and the pattern user complete the three stages of creating a user-defined pattern.

The Integration Bus developer develops the exemplar. The exemplar is fundamental to the pattern authoring process. It assumes that the exemplar is the starting point for a pattern.

The pattern author creates a pattern plug-in from the exemplar. Authoring a pattern is a design activity. Some of the pattern authoring focuses on the resources in Integration Bus, for example, defining the target properties for the pattern and configuring the user interface that is presented to the pattern user.

The pattern author ensures that the pattern users can customize the pattern as needed. If the pattern user regenerates an instance of the pattern, the pattern author also ensures that these customizations are not overwritten. The pattern author then shares the user-defined pattern with the pattern user.

The pattern user receives a user-defined pattern, customizes it, and uses it in accordance with the requirements of the organization.

## Creating a user-defined pattern

- Integration developer creates an exemplar message flow that reflects good practice for the organization
  - Exemplar pattern contains components that are needed to define the base content for the pattern
  - Use the Pattern Authoring editor in the Integration Toolkit to create and edit a pattern authoring project
- Pattern author develops pattern plug-ins and generated documentation for the pattern plug-ins
- Integration developer and pattern author test the pattern plug-ins and the generated pattern instance
- Pattern author packages the pattern plug-ins into a pattern archive
- Pattern author distributes the pattern plug-ins to individual users or a patterns community site

© Copyright IBM Corporation 2016

Figure 18-6. Creating a user-defined pattern

WM676/ZM6761.0

### Notes:

The process of developing a user-defined pattern follows these basic steps:

1. The application developer works with the *pattern author* to develop an *exemplar* pattern.
2. The pattern author reviews the exemplar pattern and determines which elements of the exemplar should be customizable by a user of the pattern. The developer and pattern author typically determine jointly which of the parameters should be customizable and how the pattern user should be able to configure them.
3. The pattern author creates a pattern authoring project in the Integration Toolkit. In the pattern authoring project, the pattern author builds the user interface with which the pattern user interacts. The user interface specifies the configurable pattern parameters, and documents the parameters so that the user knows how to set the appropriate values at pattern instantiation time. Parameters can be defined as mandatory, optional, or hidden, and can be presented to the user in a number of formats such as a list, radio button, or check box. The author can create complex transforms on parameters, including checking the content of parameter fields and doing conditional processing that is based on those values. All of these artifacts become the pattern plug-in.

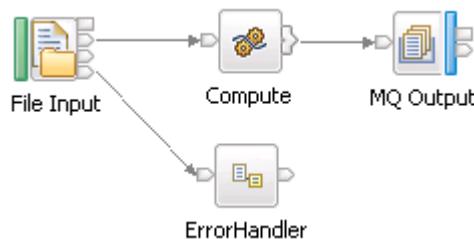
4. After defining all the pattern plug-ins, the pattern author generates the pattern plug-ins. The IBM Integration Toolkit creates a project file that contains the plug-in content.
5. After the author is satisfied with the user-defined pattern, the author creates a *pattern* archive. The archive contains the pattern plug-ins and all the elements of the user-defined pattern. It is a compressed file.
6. The pattern archive is made available to users. The archive can be sent to a user, or posted on a shared file system or on a website (such as a pattern community website) for downloading.

After the pattern archive is made available, a user can download it by clicking **Download** and **Install Pattern** from the Patterns Explorer view in the IBM Integration Toolkit. This action installs the user-defined pattern on the user workstation.

The remainder of this unit describes the detailed steps for creating a user-defined pattern.

## Creating the pattern exemplar (1 of 4)

- The *pattern exemplar* is the message flow and related components that serve as the foundation of the user-defined pattern
- Developed as an Integration Bus application



© Copyright IBM Corporation 2016

Figure 18-7. Creating the pattern exemplar (1 of 4)

WM676/ZM6761.0

### Notes:

The first step in developing a user-defined pattern is to create the exemplar pattern.

An exemplar pattern contains message flows, message maps, ESQL modules, Java classes for JavaCompute nodes, XML files, and any other components that are needed to define the base content for the pattern. The exemplar pattern should reflect good practices for the organization, since it serves as the basis for any instantiations of pattern instances.

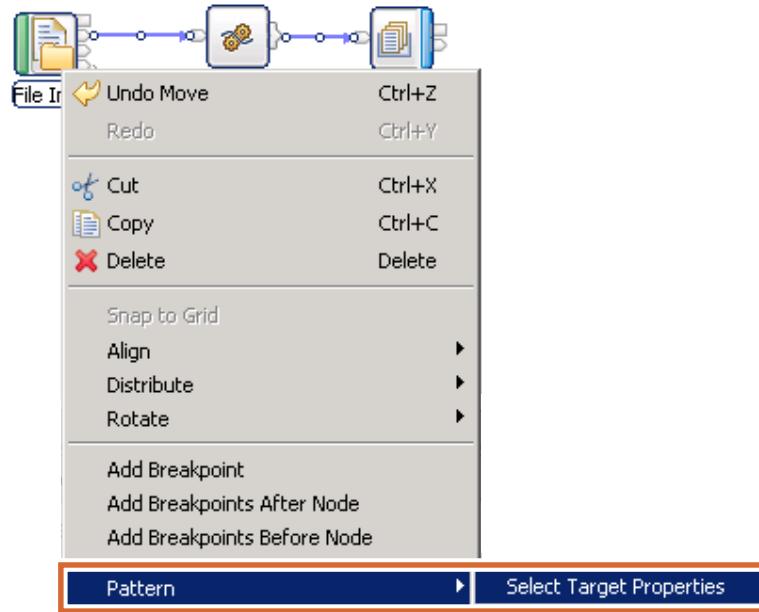
The IBM Integration Bus developer develops the exemplar, usually with the pattern author, so that the expectations of the user-defined pattern are clearly understood.

In this example, the pattern exemplar consists of a File Input node, a Compute node, an MQ Output node, and an error handling subflow.



## Creating the pattern exemplar (2 of 4)

- Select the elements that you want the pattern user to configure
  - Draw box around selected elements, right-click, and then click **Pattern > Select Target Properties**



© Copyright IBM Corporation 2016

Figure 18-8. Creating the pattern exemplar (2 of 4)

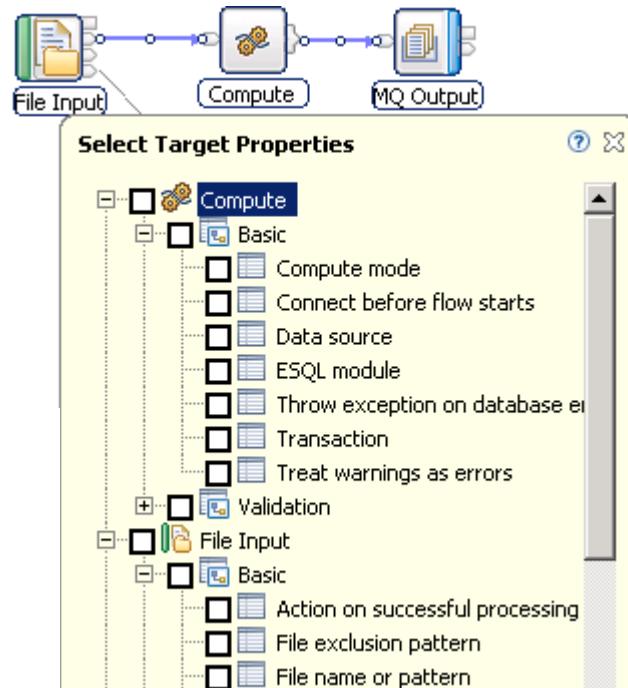
WM676/ZM6761.0

### Notes:

To define which properties the pattern user can configure at pattern instantiation, select the nodes. To select the node, draw a selection box around the nodes. The node outlines and wires turn blue. Right-click any of the nodes or wires and then click **Pattern > Select Target Properties**.

## Creating the pattern exemplar (3 of 4)

- Select the properties that the pattern user can configure



© Copyright IBM Corporation 2016

Figure 18-9. Creating the pattern exemplar (3 of 4)

WM676/ZM6761.0

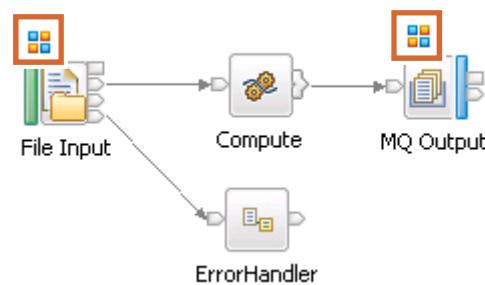
### Notes:

When you click **Select Target Properties**, a dialog box is displayed. It contains a list of all the configurable properties for the selected objects.

Select the properties that you want the pattern user to configure when the pattern is instantiated. When you are finished, close the dialog box by clicking the **X** in the upper right corner.

## Creating the pattern exemplar (4 of 4)

- Elements in pattern exemplar message flow that user can customize are flagged



© Copyright IBM Corporation 2016

Figure 18-10. Creating the pattern exemplar (4 of 4)

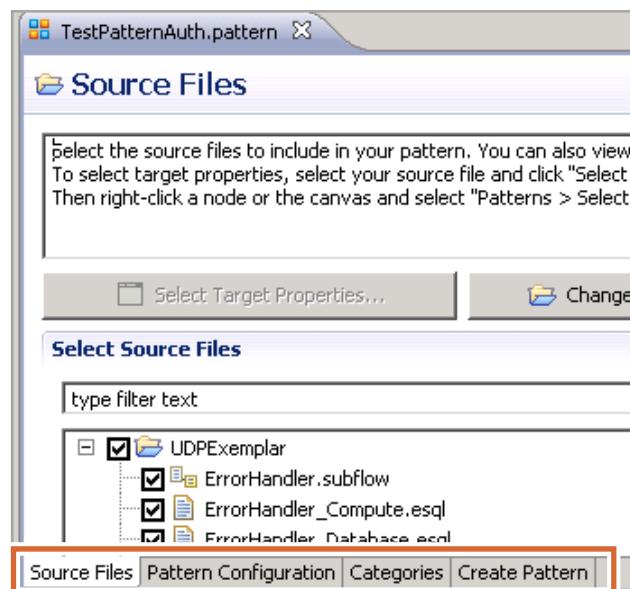
WM676/ZM6761.0

### Notes:

The exemplar has a “decorator” above the nodes in the exemplar message flow that contain user-configurable properties.

## Pattern Authoring project

1. Create a Pattern Authoring project to contain the exemplar: **File > New > Pattern Authoring Project.**
2. Select the artifacts to include in the project.
  - At minimum, include the exemplar message flow application
3. Pattern Authoring editor opens and shows the exemplar properties.
4. Follow tabs at bottom of Pattern Authoring editor to configure and create the pattern.



© Copyright IBM Corporation 2016

Figure 18-11. Pattern Authoring project

WM676/ZM6761.0

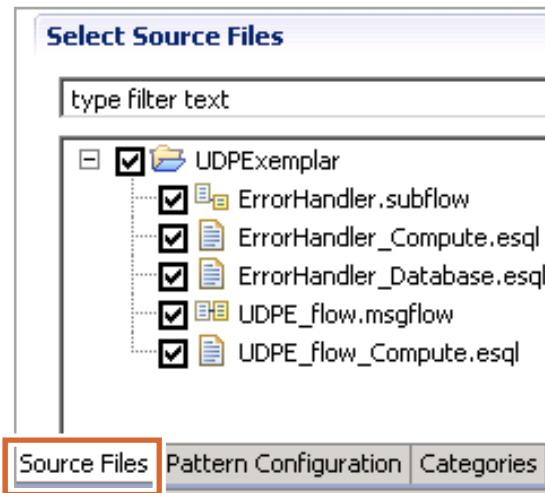
### Notes:

After the exemplar is created, the work of authoring the user-defined pattern begins. You start by creating a Pattern Authoring project in the Integration Toolkit. The Pattern Authoring editor opens.

To develop the user-defined pattern, you follow the four tabs that are shown across the bottom of the Pattern Authoring editor window.

## Selecting the source files

1. Click the **Source Files** tab.
2. Select the source files to include in the user-defined pattern.



© Copyright IBM Corporation 2016

Figure 18-12. Selecting the source files

WM676/ZM6761.0

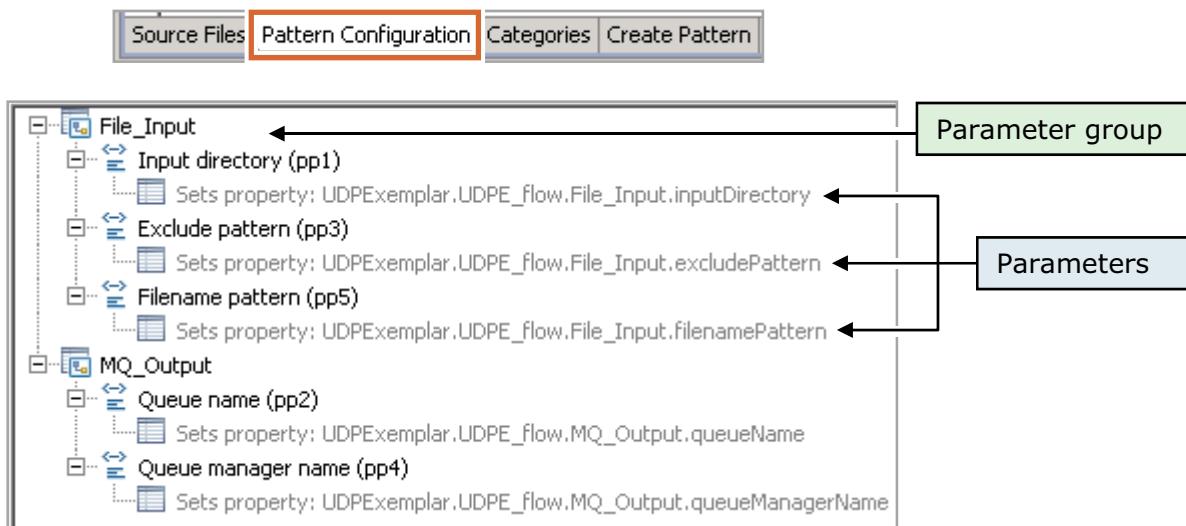
### Notes:

In the first tab section of the Pattern Authoring editor, you select the source files from the exemplar that you want to include in the final user-defined pattern. By default, all components are included.

Before you omit one or more files, consider the effect on the final project (for example, whether you want to include ESQL code for the pattern user).

## Configuring the pattern (1 of 6)

- The **Pattern Configuration** tab is where you do most of the work in defining the user-defined pattern
- Shows pattern parameters and Pattern Editor menu



© Copyright IBM Corporation 2016

Figure 18-13. Configuring the pattern (1 of 6)

WM676/ZM6761.0

### Notes:

The next step of pattern authoring is typically where you spend most of your time when developing a user-defined pattern: pattern configuration. In this step, you configure the user interface that the pattern user sees when the user-defined pattern is instantiated. This interface controls items such as:

- The names and type of parameters that the user configures
- How the parameters are grouped
- Whether parameters are mandatory or optional
- What type of widgets (user interface controls) the pattern user uses to configure the parameters when the pattern is instantiated

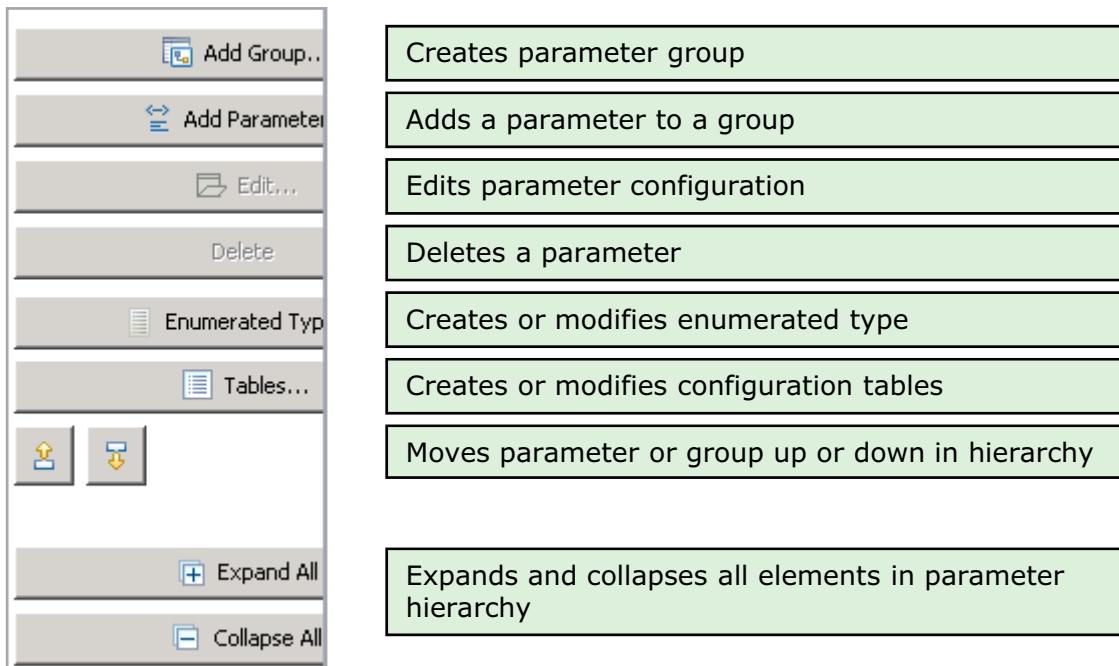
Patterns are associated with groups. By default, these groups are the nodes with which they are associated in the pattern exemplar. For example, the visual shows that **File\_Input** is a parameter group that contains the **Input\_directory**, **Exclude\_pattern**, and **Filename\_pattern** parameters. These parameters correspond to the configurable parameters that were selected for the File Input node in the pattern exemplar. You can create, delete, and modify parameters and parameter groups

in the Pattern Authoring editor. You can also associate (and disassociate) parameters with groups. You can also move parameters and groups within the displayed hierarchy.

This figure shows the left part of the Pattern Authoring editor window. The next figure shows the right part of the window.

## Configuring the pattern (2 of 6)

- Configure the parameters with the Pattern Editor menu



© Copyright IBM Corporation 2016

Figure 18-14. Configuring the pattern (2 of 6)

WM676/ZM6761.0

### Notes:

This figure shows the Pattern editor menu, which is displayed in the right part of the Pattern editor window. The pattern author uses the Pattern editor menu to configure the pattern parameters.

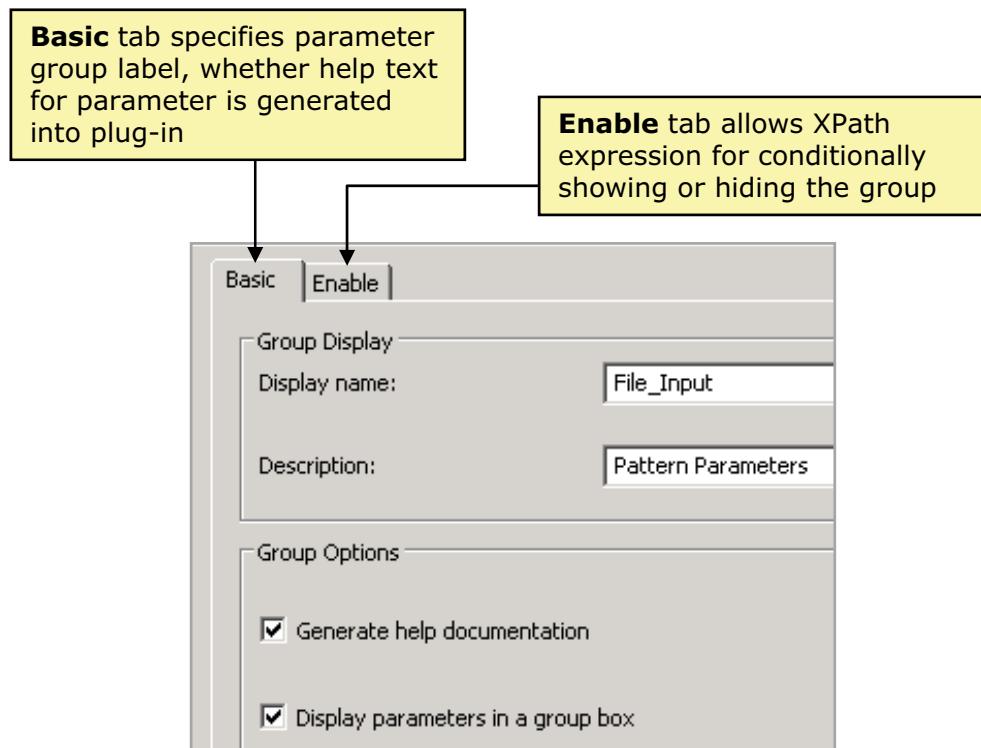
As you see on the figure, you can create parameters and parameter groups, edit parameter configurations, and move parameters and groups within the hierarchy. You can also define enumerate types and tables to use in parameter configuration.

These functions (and others) are also available by right-clicking a parameter or group.

An *enumerated type* is a set of values that a parameter accepts. It consists of a type name and a list of display names and values.

A *table* contains a collection of rows that the pattern user can configure. Each row in the table contains one or more columns of data.

## Configuring the pattern (3 of 6)



© Copyright IBM Corporation 2016

Figure 18-15. Configuring the pattern (3 of 6)

WM676/ZM6761.0

### Notes:

When you edit a parameter group, two tabs are displayed.

- Select the **Basic** tab to edit the characteristics of the parameter group, which includes the name and description. You can also control whether the help documentation for the parameter group is included with the pattern documentation.
- On the **Enable** tab, you can conditionally enable (display) or disable (hide) the parameter group, depending on the result of an XPath expression. You specify an XPath expression in the XPath Expression editor (not shown in this figure).

If the expression evaluates to true, the pattern group is enabled and is shown to the pattern user. If the expression evaluates to false, the pattern group is disabled and the pattern user cannot see it. You can select the value of any other parameters that are defined in the pattern project to use in the XPath expression. This conditional processing capability is a flexible way to control the appearance of the pattern when the pattern user instantiates the pattern.

## Configuring the pattern (4 of 6)

**Basic** Editor Transform Enable

Parameter Display

Display name:

Parameter Options

Hide the parameter Select this option to hide the parameter. Use an XPath expression to s

Mandatory parameter Select this option if the pattern requires this parameter. Mandatory parameters also d

Help Text (HTML)

Enter any HTML or text that you want to display as help text for this parameter. Do not include <html> or <head> tags because the text is inserted into an HTML file.

Field prompt:

```
<p>Describe the parameter here</p>
```

© Copyright IBM Corporation 2016

Figure 18-16. Configuring the pattern (4 of 6)

WM676/ZM6761.0

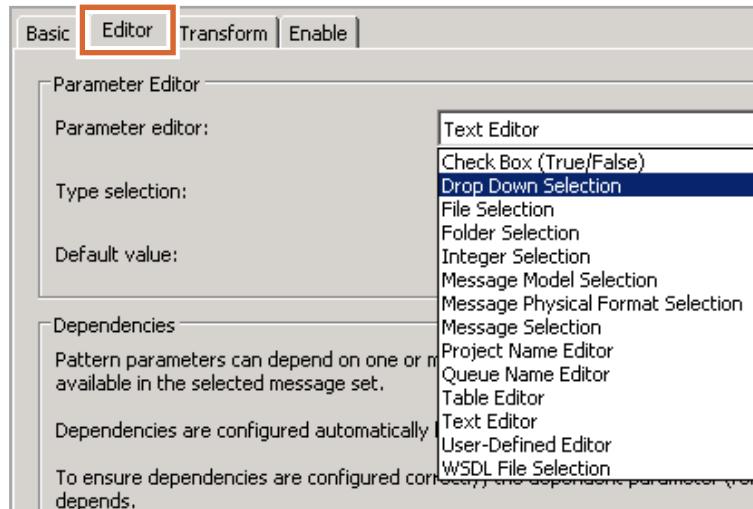
### Notes:

When you select a parameter to edit, four tabs are presented in the editor: **Basic**, **Editor**, **Transform**, and **Enable**.

On the **Basic** tab, you set parameter attributes such as the display name, whether the parameter is mandatory, and whether the parameter is hidden. You also set the field prompt (which is displayed in the parameter field itself at instantiation time). Also, you can create help text for the parameter by writing HTML. When you format the help text, a secondary window shows you how the help text is rendered to the pattern user.



## Configuring the pattern (5 of 6)



- **Editor tab:**

- Sets the type of parameter and the control widget that the user uses to alter the parameter
- Selects the most appropriate type automatically, but you can override
- Specifies any dependency that this parameter has on any others

© Copyright IBM Corporation 2016

Figure 18-17. Configuring the pattern (5 of 6)

WM676/ZM6761.0

### Notes:

The **Editor** tab is where you control the specifics of the user interface for the parameter as it is shown to the pattern user. These attributes include the type of editor that is used to modify the field and provide an optional default value for the parameter (if the parameter type allows it).

The Parameter editor is based on the parameter type. In most cases, the Pattern Authoring editor selects the most appropriate parameter editor, depending on the parameter type, but you can override this value.

Some of the parameter editors are listed here (this list is not complete):

- Check box (true/false)
- Drop-down list
- File selection menu
- Folder selection menu
- Message model selection
- Project name editor
- Queue name editor
- Text editor
- WSDL file selector menu

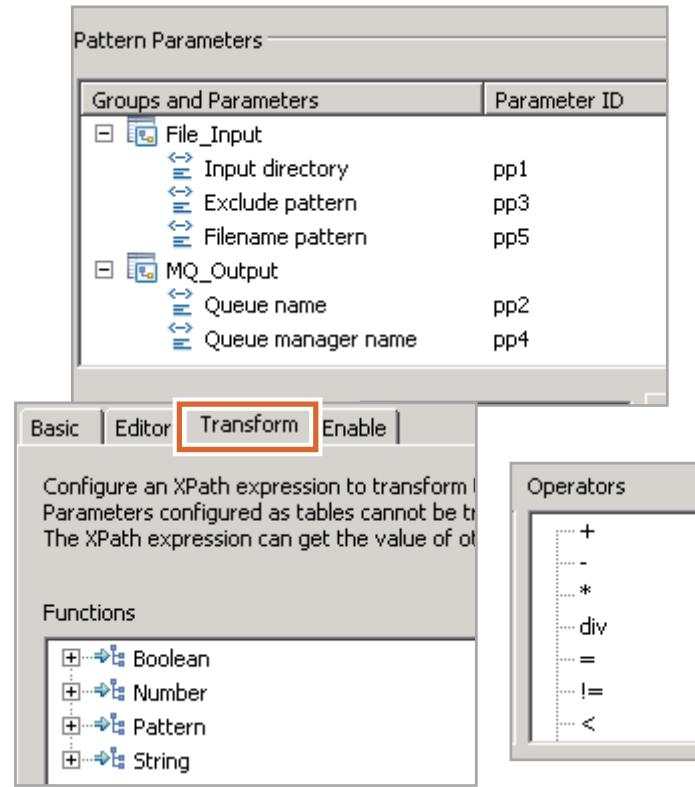
Most of these editors contain configurable properties to assist the pattern user. For example, if you choose the **File selection menu** editor, you are prompted to provide a list of file extensions that are displayed in the field prompt for the pattern user.

You can also create user-defined editors that are specific to the parameter type.

On the **Editor** tab, you can also view dependencies between parameters. For example, the editor that is used for a parameter can depend on the editor that is used for a second parameter, or on its values. The Pattern Authoring editor manages these dependencies for you, but the dependencies are listed here for your reference.

## Configuring the pattern (6 of 6)

- Use the **Transform** tab to define the XPath expression to transform the value of the parameter
- Query the value of other parameters by using the `getValue()` expression



© Copyright IBM Corporation 2016

Figure 18-18. Configuring the pattern (6 of 6)

WM676/ZM6761.0

### Notes:

You use the **Transform** tab to define expressions to transform the value of a parameter.

When the pattern user creates the pattern instance (by clicking **Generate** in the Pattern Explorer), the XPath expression is evaluated and the transform occurs. The transform can evaluate the value of other parameters by using the `getValue()` XPath expression. An XPath expression builder is included on this tab to help you construct the appropriate XPath expressions and to evaluate them for testing.

If you are using tables as part of the parameter configuration, you use Java code instead of XPath expressions to evaluate the table values.

You use the **Enable** tab to create an XPath expression that controls whether the parameter is visible to the pattern user in the Pattern Instance editor.

## Setting the Patterns Explorer category

- Use the **Categories** tab to set the location in the Patterns Explorer for the new pattern
  - Drag to the location you want
  - Optionally, create a category for pattern

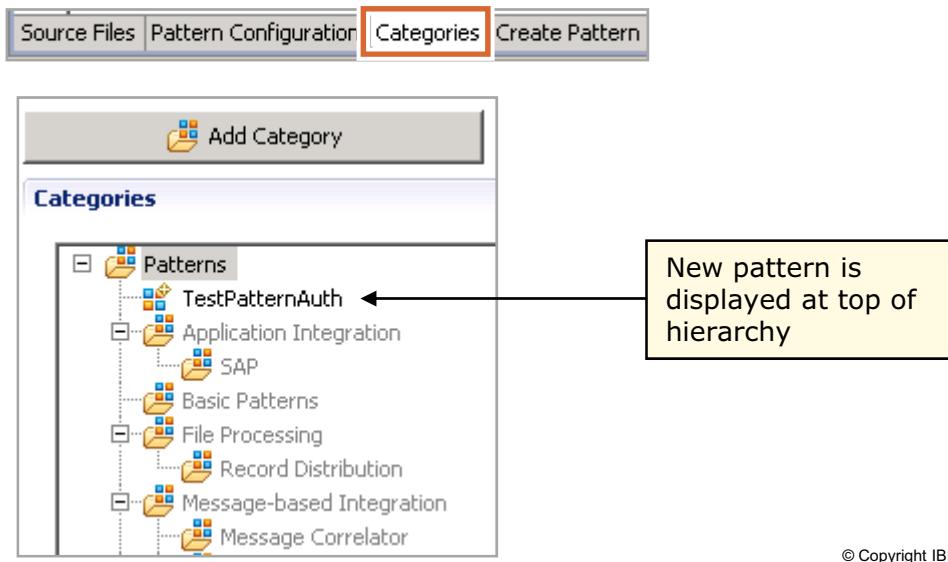


Figure 18-19. Setting the Patterns Explorer category

WM676/ZM6761.0

### Notes:

Before you generate the pattern plug-ins, you must define where the new pattern is displayed in the Patterns Explorer in the Integration Toolkit. To do so, select the **Categories** tab, and then drag the new pattern name to the location in the pattern hierarchy that you want. You can also create a category to contain the pattern by clicking **Add Category**. You can add the category at any location in the hierarchy.

Normally you set the pattern location after you customize all the parameters and parameter groups, but you can do it any time before you generate the pattern plug-ins.



## Creating the pattern (1 of 2)

- Use the **Create Pattern** tab to:
  - Create a file that contains the pattern plug-ins
  - Test the pattern, and debug if necessary
  - Create the pattern archive that contains the distributable pattern package

Plug-in Information	
	Configure the unique identifier for your pattern plug-in.
Pattern name:	TestPatternAuth
Plug-in ID:	MyBigCompany.TestPatternProject
Version:	1.4.0
Provider:	My Big Company
Description:	Package contains new pattern

Set pattern plug-in attributes for distribution

© Copyright IBM Corporation 2016

Figure 18-20. Creating the pattern (1 of 2)

WM676/ZM6761.0

### Notes:

You do the final steps of creating a user-defined pattern in the **Create Pattern** tab. These steps include:

- Setting the attributes for the pattern plug-ins file.
- Generating a file that contains the pattern plug-ins.
- Testing and debugging the pattern.
- Creating the pattern archive file. This file contains the user-defined pattern in distributable form.

As you saw with the other aspects of pattern development, the Pattern Authoring editor guides you through these last steps by presenting a window that contains controls for you to use.

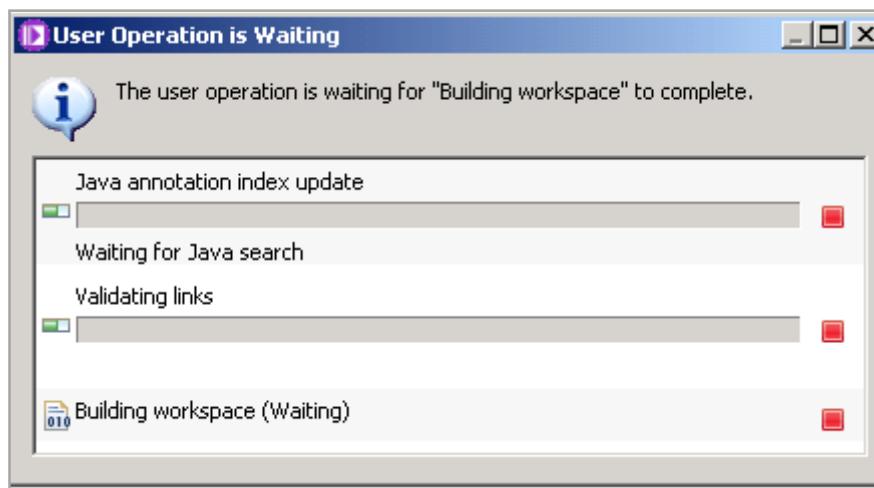
On the **Create Pattern** tab, you set attributes for the plug-ins file. These attributes are packaged with the plug-ins themselves. The information that you provide on this page typically follows your development standards for nomenclature, standards, and version numbers.

## Creating the pattern (2 of 2)

- Before testing the pattern, you must create the pattern plug-ins
- Click **Create Pattern Plug-ins**

 Create Pattern Plug-ins

- Can be a resource-intensive process, depending on the size and complexity of the pattern project



© Copyright IBM Corporation 2016

Figure 18-21. Creating the pattern (2 of 2)

WM676/ZM6761.0

### Notes:

Before you can test a pattern, you must generate a file that contains the pattern plug-ins. To do so, on the **Create Pattern** tab, click **Create Pattern Plug-Ins**.

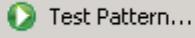
Depending on the size and the complexity of the pattern project, this operation can take several minutes and use much of the workstation processor power.

The file that contains the plug-ins is made available for testing.



## Testing the pattern

- Testing a new pattern is an essential part of developing it
- When testing, the pattern author sees what the pattern user sees
- Test from the perspective of the user of the new pattern
  - Are the correct parameters included?
  - Do the prompts make sense?
  - Is the help text meaningful?
  - Does the generated message flow work as intended?

1. Click **Test Pattern**. 
2. A second copy of the Integration Toolkit opens. Specify a different workspace for testing.
3. From the **Quick Starts** pane, click **Start from Patterns**.

© Copyright IBM Corporation 2016

Figure 18-22. Testing the pattern

WM676/ZM6761.0

### Notes:

A critical aspect of creating a user-defined pattern is testing it. When you test a pattern, a second copy of the Integration Toolkit opens. It includes the pattern plug-in that you created in the previous step.

Select a workspace when the toolkit opens. You cannot use the same workspace that is open for the Pattern Authoring editor (in the original instance of the toolkit).

Open the Patterns Explorer from the **Quick Start** pane by clicking **Start from Patterns**.

The new pattern is displayed in the Patterns Explorer in the location you set in the previous steps. Select the pattern and begin testing.

As you test, consider yourself to be the pattern user. Some considerations for testing include:

- Are the correct parameters available for the user to configure?
- Do the configuration steps make sense, and are they presented in a logical progression?
- Are the correct editors used?
- Are the default values appropriate?

- If parameters and parameter groups were set to appear conditionally, do they appear (or not appear) as expected?
- Are the documentation for the pattern and the parameters correct?
- Does the generated message flow work as intended?

Depending on your development standards, you might consider having someone else test the new pattern, or forward it to your testing team.

When you are finished testing, close the second copy of Integration Toolkit.

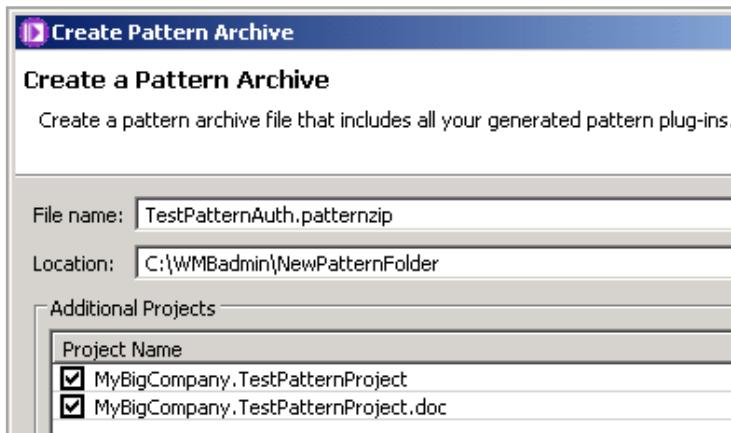
## Creating the pattern archive

Pattern archive is the compressed file that is used to distribute the new pattern to users

1. Click **Create Pattern Archive**.



2. You are prompted for the components to include in the pattern archive and the archive file name and location.



© Copyright IBM Corporation 2016

Figure 18-23. Creating the pattern archive

WM676/ZM6761.0

### Notes:

After you complete all development and testing for the new pattern, you can package it for distribution by creating a pattern archive. A pattern archive is a compressed file, which contains all the artifacts that are needed to deploy the pattern to other Integration Toolkit installations.

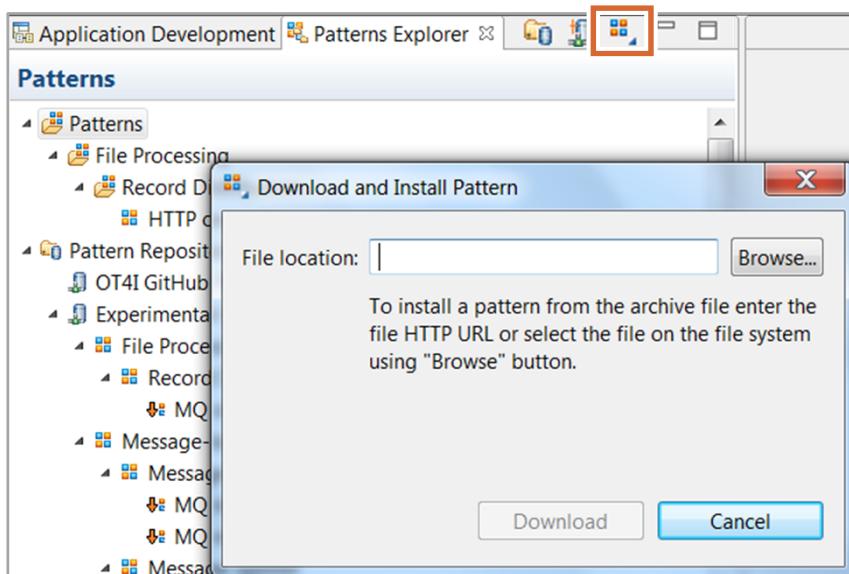
To create the pattern archive, click **Create Pattern Archive** on the **Create Pattern** tab. You are prompted for the name and location for the archive file, and the components to include in it.

Like the generation process for the pattern plug-ins, creating the pattern archive can be resource-intensive and can take an extended amount of time to complete.

After the file is generated, you can distribute it according to the policies of your organization.

## Installing the user-defined pattern

- A user can download and install the new pattern from the **Patterns Explorer** in the IBM Integration Toolkit
- Click the **Download and install** icon and then specify the download location



© Copyright IBM Corporation 2016

Figure 18-24. Installing the user-defined pattern

WM676/ZM6761.0

### Notes:

To install the user-defined pattern, you use the Patterns Explorer. Click **Download**, and then select the location of the pattern archive (.patternzip) file. When the file is located, the IBM Integration Toolkit automatically extracts the components and adds the pattern to the Patterns Explorer.

You can also uninstall a pattern that you installed. You use the Integration Toolkit to uninstall.

1. From the toolbar, click **Help > About IBM Integration Toolkit - IBM Integration Bus**.
2. Click **Installation Details**.
3. Click **Installed Software**.
4. Browse to and select the pattern feature.
5. Click **Uninstall**.
6. Select the pattern feature to uninstall, and then click **Finish**. The pattern feature is uninstalled.
7. You can either restart the workspace, or click **Apply Changes**.
8. Click **OK** to close the **About IBM Integration Toolkit - IBM Integration Bus** window.

## Unit summary

Having completed this unit, you should be able to:

- Construct and extend a user-defined pattern
- Create a pattern authoring project
- Build pattern plug-ins
- Package and distribute pattern plug-ins
- Install a pattern archive

© Copyright IBM Corporation 2016

---

Figure 18-25. Unit summary

WM676/ZM6761.0

### Notes:

## Checkpoint questions

1. True or false: You must use a separate IBM Integration Toolkit instance to create user-defined patterns.
2. True or false: One of the benefits of using patterns is that it helps to promote reuse of well-designed code that is based on good practices.
3. Place these pattern development steps into the correct order:
  - a. Test the pattern plug-ins
  - b. Create a pattern authoring project
  - c. Develop a pattern exemplar
  - d. Configure the pattern authoring project
  - e. Distribute the pattern archive
  - f. Generate the pattern plug-ins
  - g. Create the pattern archive

© Copyright IBM Corporation 2016

Figure 18-26. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
- 2.
- 3.



## Checkpoint answers

1. True or false: You must use a separate IBM Integration Toolkit instance to create user-defined patterns.  
Answer: **False**
2. True or false: One of the benefits of using patterns is that it helps to promote reuse of well-designed code that is based on good practices.  
Answer: **True**
3. Place these pattern development steps into the correct order:
  - a. Test the pattern plug-ins
  - b. Create a pattern authoring project
  - c. Develop a pattern exemplar
  - d. Configure the pattern authoring project
  - e. Distribute the pattern archive
  - f. Generate the pattern plug-ins
  - g. Create the pattern archive  
Answer: **c, b, d, f, a, g, e**

© Copyright IBM Corporation 2016

Figure 18-27. Checkpoint answers

WM676/ZM6761.0

### Notes:

# Unit 19. Extending IBM Integration Bus

## What this unit is about

In this unit, you learn how IBM Integration Bus functions can be extended with other IBM products, industry content packs, and enterprise information systems.

## What you should be able to do

After completing this unit, you should be able to:

- Describe how IBM Integration Bus integrates with other IBM products such as IBM WebSphere Enterprise Service Bus and IBM DataPower Appliances
- Describe how IBM Integration Bus can interact with enterprise information systems

## How you will check your progress

- Checkpoint

## References

IBM Knowledge Center for IBM Integration Bus V10



## Unit objectives

After completing this unit, you should be able to:

- Describe how IBM Integration Bus integrates with other IBM products such as IBM WebSphere Enterprise Service Bus and IBM DataPower Appliances
- Describe how IBM Integration Bus can interact with enterprise information systems

© Copyright IBM Corporation 2016

---

Figure 19-1. Unit objectives

WM676/ZM6761.0

### Notes:

## IBM Integration Bus EIS adapter nodes

- WebSphere Adapters are delivered ready for use as built-in nodes communicate with enterprise information systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards
  - Simplifies management and improves performance for key integration scenarios
  - JCA-based WebSphere Adapters
- Support for inbound and outbound scenarios
  - Message-to-EIS and EIS-to-message scenarios
  - Input and output nodes integrate with all built-in IBM Integration Bus nodes
- Integration Toolkit metadata discovery
  - Simplifies EIS query for key data structure discovery
  - Simplifies and accelerates the generation of message sets
  - Incremental discovery to add only new definitions to an existing project



SAP Input



Siebel Input



PeopleSoft Request

© Copyright IBM Corporation 2016

Figure 19-2. IBM Integration Bus EIS adapter nodes

WM676/ZM6761.0

### Notes:

WebSphere adapter nodes communicate with enterprise information systems (EIS). The adapters are based on the JCA V1.5 WebSphere adapter foundation classes, which are hosted in the IBM Integration Bus natively as nodes. Using nodes in Integration Bus simplifies the management of the adapters and improves performance for key integration scenarios because no outside components must be started or stopped.

Support is provided for the major EIS systems such as SAP, Siebel, PeopleSoft, and JD Edwards. Also provided is a sample adapter named Twineball. The TwineBallInput and TwineBallRequest nodes are sample nodes with their own sample EIS. You can use the TwineBall nodes to see how adapters nodes work without having to connect to an external EIS.

Each adapter requires a separate license entitlement to deploy message flows to an integration node for any purpose other than unit testing on the developer's workstation.

The WebSphere Adapters support two modes of communication: inbound and outbound.

Inbound is where an event is generated on the EIS and the adapter responds to the event by sending a message to IBM Integration Bus. The WebSphere Adapters input nodes support inbound

communication. When the EIS sends an event to the adapter, a message is propagated from the WebSphere Adapters input node.

Outbound is where the Integration Bus uses the adapter to send a request to the EIS. The WebSphere Adapters request nodes support outbound communication. When a message is propagated to the WebSphere Adapters request node, the adapter sends a request to the EIS.

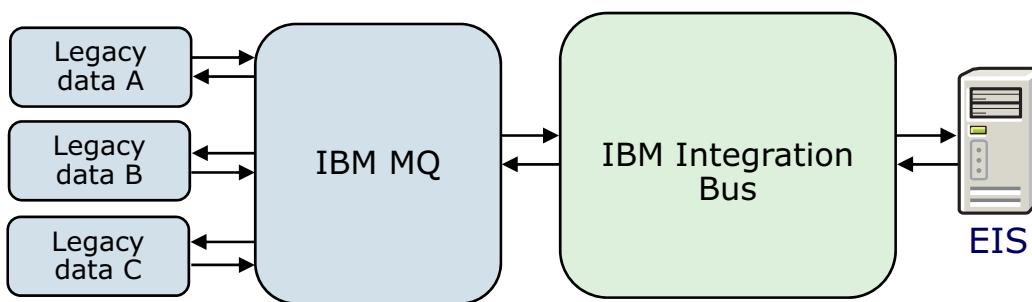
Enterprise Metadata Discovery is a specification that you can use to examine an EIS and get details of business object data structures and APIs. In Integration Bus, you use the **Adapter Connection** wizard to examine an EIS. The definitions are then stored as XML schema by default. The wizard is used to quickly build components that can access the EIS.

Because the adapters use the Integration Bus tree, the bitstream has no unnecessary serialization, thus providing high-performance access.

See <http://www.ibm.com/software/integration/wbiadapters/apps/> for information about adapters and supported versions.

## Scenario 1: Legacy data interface to an EIS

- An internal infrastructure communicates between different components by using COBOL copybook structures over IBM MQ, requiring that a new EIS is integrated into this infrastructure



© Copyright IBM Corporation 2016

Figure 19-3. Scenario 1: Legacy data interface to an EIS

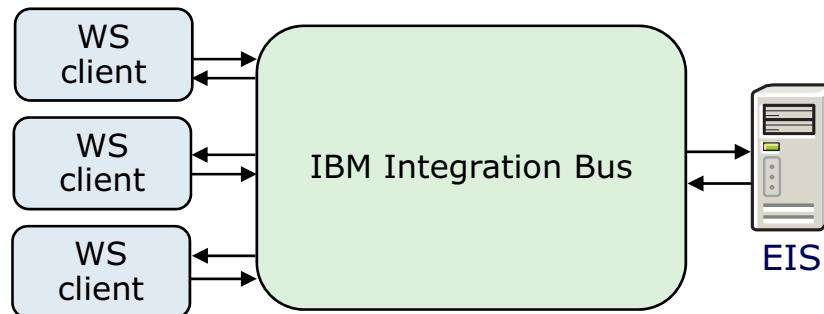
WM676/ZM6761.0

### Notes:

In this scenario, Integration Bus can be used to provide the interface between legacy data applications and an EIS.

## Scenario 2: Web services interface to an EIS

- Provide capability in the EIS through a web service interface by using the HTTP/SOAP nodes with the EIS nodes



© Copyright IBM Corporation 2016

Figure 19-4. Scenario 2: Web services interface to an EIS

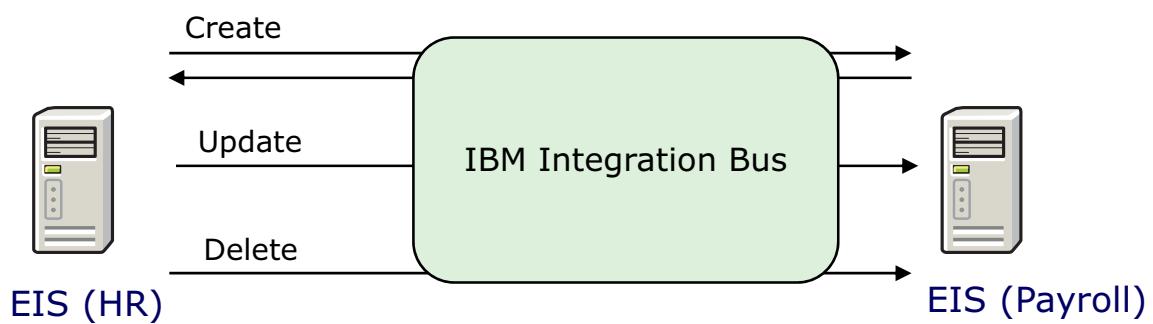
WM676/ZM6761.0

### Notes:

In this scenario, Integration Bus can be used to make the EIS accessible through web services by using HTTP and SOAP nodes.

## Scenario 3: Synchronizing enterprise information systems

- Ensuring that HR and payroll systems contain identical information (for example, after a change of address)



© Copyright IBM Corporation 2016

Figure 19-5. Scenario 3: Synchronizing enterprise information systems

WM676/ZM6761.0

### Notes:

In this scenario, Integration Bus can be used to synchronize two different EIS such as PeopleSoft and Siebel.

## Significant extensions for packaged applications

- Siebel and PeopleSoft operational reconfiguration
  - Eases promotion of Siebel and PeopleSoft message flows through the test, QA, and production lifecycles
  - Configurable service provides the reconfiguration of key adapter node properties, including host name, client ID, system number, user ID, and password
  - Supports wholesale replacement of adapter connection
- SAP
  - Single program ID to allow different flows to handle multiple IDOCs without disruption
  - SCI support with the SAP Reply node
  - High availability for SAP Input nodes
  - BAPI commits wait processing
- General
  - Incremental deployment to easily add new definitions to existing deployments
  - User-defined operations in addition to create, read, update, and delete operations

© Copyright IBM Corporation 2016

Figure 19-6. Significant extensions for packaged applications

WM676/ZM6761.0

### Notes:

IBM Integration Bus includes significant extensions for SAP, Siebel, PeopleSoft, and JD Edwards. It includes fully configurable services for user IDs, passwords, servers, and all properties for connectivity to the engineering resource planning (ERP) system. The configurable services can be modified in the IBM Integration web user interface and command interface.

Integration Bus enhancements allow communication with a single SAP program ID and provide synchronous RFC support for the SAP reply node. Integration Bus can also support high availability for the SAP node. For example, suppose that an SAP input node is in a message flow and another node is running in another integration node. The state is stored in the multi-instance queue manager, so that if one instance fails, the other can take over.

Other functions include incremental discovery and incremental deployment. If you have an adapter project with 50 adapter definitions, you can go back and add more definitions without reprocessing the original 50. Incremental discovery and deployment also allow multiple adapters to use the same connection configuration.

## Business activity monitoring with IBM Business Monitor

- Make the business easier by allowing evidence-based decision making
- Decision-makers need key performance indicators (KPIs)
  - Best source of KPIs is the applications that run the business
  - Send the monitoring events to a monitoring application for analysis and display
- IBM Integration Bus is an ideal source for monitoring events because it offers
  - High-performance XML handling
  - Flexibility
  - Visibility of data in any format
- To monitor message flows from IBM Integration Bus, generate a monitor model in the IBM Business Monitor development toolkit

© Copyright IBM Corporation 2016

Figure 19-7. Business activity monitoring with IBM Business Monitor

WM676/ZM6761.0

### Notes:

Business activity monitoring is the aggregation, analysis, and presentation of real-time information about activities. You can extract data about the flow, in real time, and decide based on the data.

With IBM Integration Bus, you can identify events and audit the data.

 WebSphere Education 

## Monitoring information for IBM Business Monitor

- Export message flow from Integration Bus
- Import monitoring information to start the Generate Monitor Model wizard and create a model that contains:
  - Inbound events for each event source that is defined in the message flow
  - localTransactionId defined as a key
  - More metrics and KPIs are available by selecting templates
- A log file is created in an Integration Bus message flow project to show output from a generated process



© Copyright IBM Corporation 2016

Figure 19-8. Monitoring information for IBM Business Monitor

WM676/ZM6761.0

### Notes:

In Integration Bus, a monitor model is created by using a wizard.

- Inbound event names are taken from the event source address name.
- Inbound events for IBM Integration Bus transaction event sources are created at the flow level in the model.
- Event groups are created for each of the nodes where the IBM Integration Bus terminal event sources are defined. The inbound events for these IBM Integration Bus terminal events are then stored in the event groups.
- Inbound events for subflow event sources contain the parent and subflow names as a prefix. For example, if the message flow ParentFlow contains the subflow that is named SubFlow that has an event source MQInput.terminal.out, the inbound event has the name ParentFlow.SubFlow.MQInput.terminal.out.

The **Synchronize with Application** option in the Integration Toolkit updates the created model with changes in the message flow event source definitions.



## Integration requirements for IBM Business Monitor

- Always define transaction events (`transaction.Start`, `transaction.End`, and `transaction.Rollback`) on message flows so that Business Monitor can identify the start and end of a monitoring context
- Create an event part with a supported type
  - Business Monitor Toolkit does not support local elements with anonymous types
  - Business Monitor Toolkit does not support creating metrics of type `xs:anyType`
- Install and configure a message-driven bean in IBM Business Monitor
  - Message-driven bean, which runs in WebSphere Application Server, subscribes to the event topic and writes the events that match its subscription to the CEI repository as events that conform to the Common Base Event specification

© Copyright IBM Corporation 2016

Figure 19-9. Integration requirements for IBM Business Monitor

WM676/ZM6761.0

### Notes:

IBM Business Monitor must be able to identify the start and end of a monitoring context. Always define transaction events (`transaction.Start`, `transaction.End`, and `transaction.Rollback`) on message flows that Business Monitor monitors.

The IBM Business Monitor toolkit does not support local elements with anonymous types. Therefore, the export monitoring information option does not generate an event part for event payload XPath queries that resolve to an element of this type. You see a warning message in the report log.

The IBM Business Monitor toolkit does not support creating metrics of type `xs:anyType`. If an XPath expression in your event payload resolves to an element of type `xs:anyType`, the export monitoring information option creates an event part of this type. You cannot create a metric of this type in the IBM Business Monitor toolkit. Create an event part with a supported type.

If you include unmodeled data, the export monitoring information option cannot type the data. It assigns a type of string to the data.

The option to export monitoring information does not support XPath queries that contain wildcards.

## Conversion from WebSphere Enterprise Service Bus

- Preserves structural wiring between primitives of a mediation flow
  - No minimum version requirement for WebSphere Enterprise Service Bus source assets
- Built-in converters for conversion of the following entities are included:
  - Export bindings
  - Synchronous import bindings
  - Most mid-flow primitive types
- Automatic conversion of the following entities is supported:
  - “Service Invoke” primitive
  - Modules with more than one export, component, or interface
  - Modules with plain old Java objects (POJOs)
  - Subflows

© Copyright IBM Corporation 2016

Figure 19-10. Conversion from WebSphere Enterprise Service Bus

WM676/ZM6761.0

### Notes:

IBM Integration Bus represents IBM's strategic ESB offering and is the successor product for existing clients of both WebSphere Message Broker and WebSphere Enterprise Service Bus.

Integration Bus provides tools that facilitate the conversion of existing WebSphere Enterprise Service Bus assets so that they can run on IBM Integration Bus.

- WebSphere Enterprise Service Bus Project Interchange files can be imported and viewed.
- Common flow primitives are converted automatically while maintaining the flow structure.
- A task list describes the remaining manual tasks that must be completed.
- Flows are created and can be modified and deployed.

The conversion tool is built upon an extensible framework, enabling further enhancements that reduce the number of manual tasks required.



## Example WebSphere Enterprise Service Bus conversion

Configure global conversion options. Add extensions for those resources for which you want to use your own conversion code.

**Conversion Result**  
Specify how the conversion result should be recorded.  
 Merge new conversion results with the results from previous runs of this conversion session

**Mediation Primitive Converters**  
Each mediation primitive will be converted to a message flow node or subflow. You can supply your own converter to convert mediation primitive to see information on its usage analysis.

Mediation Primitive	Convert to	Usage	Converter class
InputResponse	Reply (for example SOAPReply)	StockQuote_MediationFlow.component	Built-in converter
MessageElementSetter	JavaCompute	StockQuote_MediationFlow.component	Built-in converter
MessageFilter	Route	StockQuote_MediationFlow.component	Built-in converter
MessageLogger	Subflow placeholder	StockQuote_MediationFlow.component	Placeholder converter
XSLTransformation	Map	StockQuote_MediationFlow.component	Built-in converter

**Task List**

1. Export the WebSphere Enterprise Service Bus project
2. Import mediations into the IBM Integration Toolkit
3. Right-click the “convert” task to start a conversion
4. Follow the guided editor to generate resources
5. Task List identifies the remaining manual steps

© Copyright IBM Corporation 2016

Figure 19-11. Example WebSphere Enterprise Service Bus conversion

WM676/ZM6761.0

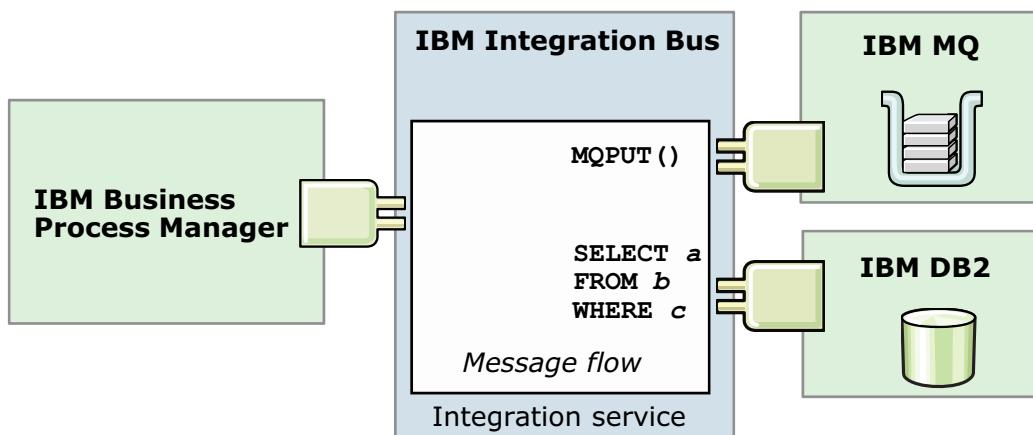
### Notes:

The steps for converting a WebSphere Enterprise Service Bus project to an IBM Integration Bus project are:

1. Export the WebSphere Enterprise Service Bus project.
2. Import mediations into the IBM Integration Toolkit.
3. Right-click “convert” to start conversion.
4. Follow guided editing to generate resources.
5. Complete the tasks in the Task List.

## Connectivity to IBM Business Process Manager

- Use IBM Integration Bus with IBM Business Process Manager integration services
- IBM Business Process Manager developer can use it to concentrate on human task processing by delegating the connectivity configuration to the Integration Bus developer
- Uses integration services



© Copyright IBM Corporation 2016

Figure 19-12. Connectivity to IBM Business Process Manager

WM676/ZM6761.0

### Notes:

You can use IBM Process Designer to design IBM Business Process Manager integration services, including input and output business objects, and use these definitions to generate services and operations in IBM Integration Bus.

From IBM Business Process Manager, a process designer can specify the input data that is sent to a system task and the response that is expected from the task. This information is passed to the Integration Bus development team. The Integration Bus development team implements a service that accepts the defined request and delivers the expected response.

## Business Process Manager Advanced nodes in IBM Integration Bus

- SCA Input and SCA Reply nodes
  - Receive inbound messages from the Business Process Manager Advanced
  - Support web services and IBM MQ bindings
- SCA Request node
  - Sends outbound synchronous request messages to the Business Process Manager Advanced
- SCA Asynchronous Request and SCA Asynchronous Response nodes
  - Send outbound asynchronous request messages to the Business Process Manager Advanced

© Copyright IBM Corporation 2016

Figure 19-13. Business Process Manager Advanced nodes in IBM Integration Bus

WM676/ZM6761.0

### Notes:

IBM Business Process Manager Advanced includes three layers:

1. The SOA of IBM Business Process Manager Advanced provides both uniform invocation and data representation programming models, and monitoring and management capabilities for applications that run on IBM Business Process Manager Advanced.
2. Supporting services in IBM Business Process Manager address a number of transformation challenges for connecting components and external artifacts.
3. All integration artifacts that run on IBM Business Process Manager (for example, business processes, business rules, and human tasks) are represented as service components with well-defined interfaces.

The SCA Input node allows an IBM Business Process Manager Advanced SCA Import component to use IBM Integration Bus as an SCA endpoint.

The SCA Reply node sends a response message from the integration node back to the originating Process Server SCA client in response to a message received by an SCA Input node.

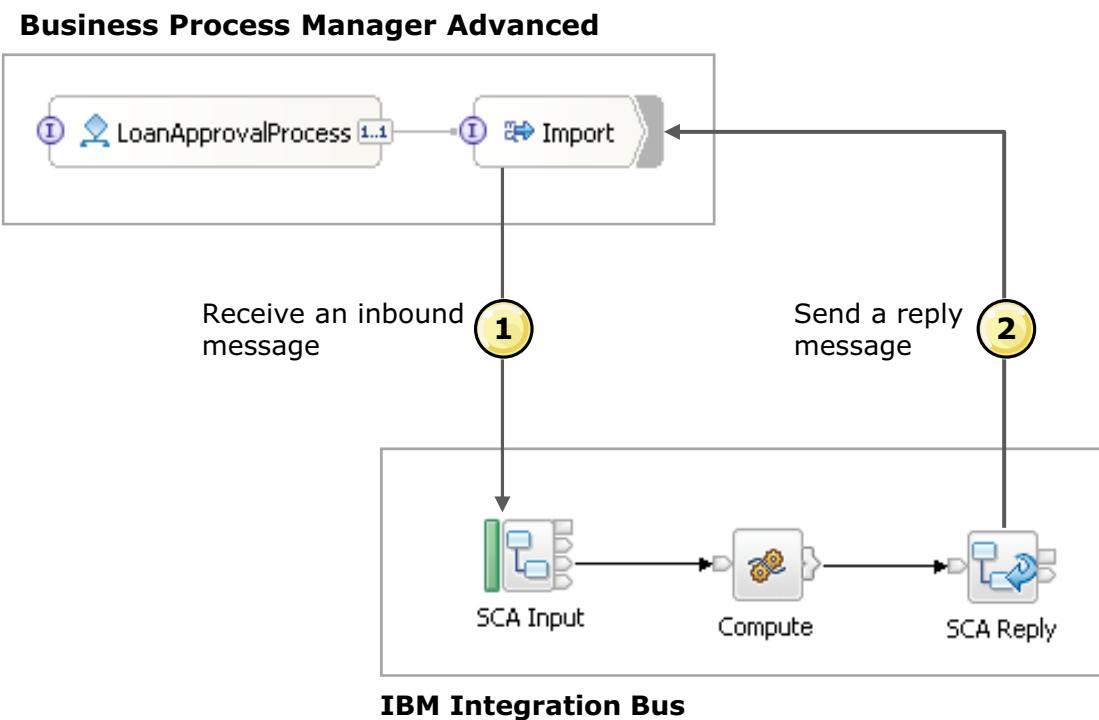
The SCA Request node sends synchronous requests from the integration node, allowing the integration node to participate in synchronous message exchange patterns with a Process Server SCA Export component.

The SCA Asynchronous Request node sends asynchronous requests from the integration node, allowing the integration node to participate in asynchronous message exchange patterns with an IBM Business Process Manager Advanced SCA Export component.

The SCA Asynchronous Response node allows the integration node to receive the response to a previous asynchronous request made from an SCA Asynchronous Request node.

For each node, the message the integration node receives can be a SOAP/HTTP or an IBM MQ message, depending on the binding that the node used.

## Example: Processing an inbound message



© Copyright IBM Corporation 2016

Figure 19-14. Example: Processing an inbound message

WM676/ZM6761.0

### Notes:

The message flow in the figure uses an SCA Input node to receive a request from SCA components that are running on IBM Business Process Manager Advanced.

1. The SCA Import component in IBM Business Process Manager Advanced sends the inbound requests to the Integration Bus SCAInput node, which uses dynamic terminals to process the message body. The operations that are defined in the WSDL interface file determine these terminals. An incoming message is routed to the appropriate terminal as determined by the target operation.

A ReplyIdentifier on the SCA Input node is set in the message context and the local environment. The value must be preserved throughout the flow, especially if the SCA Input and SCA Reply nodes are in different flows so that the reply can get back to the originating client.

2. An SCA Reply node sends a reply to the IBM Business Process Manager Advanced business process.

If the user does not have an integration node SCA definition with which to configure the node, the SCA Input and SCA Reply nodes cannot be used.



## Wizards for SCA components

- SCA Importer imports an SCA import or an SCA export from a WebSphere Integration Developer project interchange file
- SCA Generator creates an IBM Integration Bus SCA definition, used to configure SCA nodes, from a message set
- SCA Exporter exports an SCA import or SCA export from an IBM Integration Bus SCA definition for import into WebSphere Integration Developer

© Copyright IBM Corporation 2016

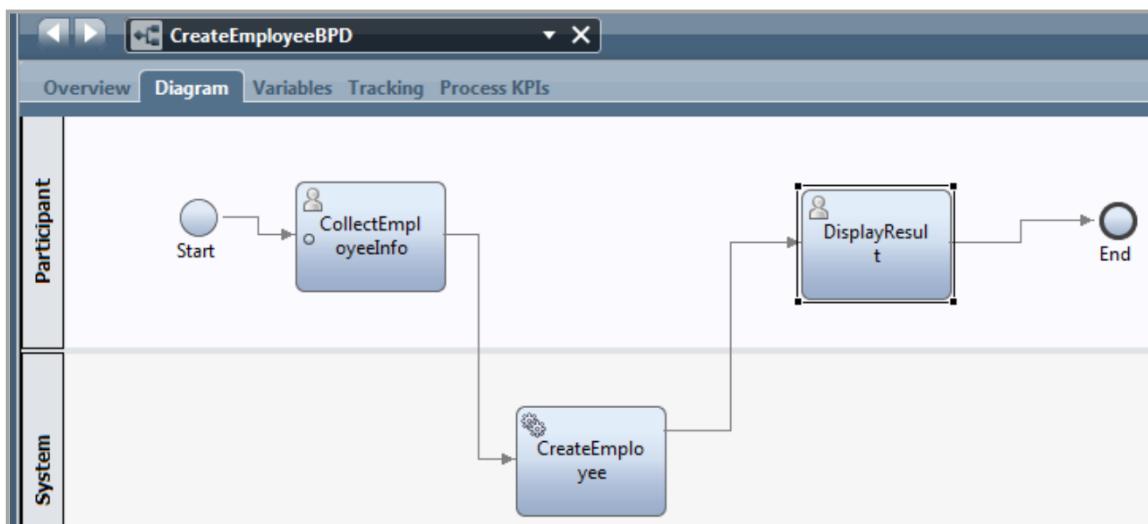
Figure 19-15. Wizards for SCA components

WM676/ZM6761.0

### Notes:

The figure summarizes the wizards that are available to help integrate an IBM Integration Bus message flow with IBM Business Process Manager Advanced.

## Synergy with Business Process Manager



- IBM Integration Bus provides a powerful connectivity layer for IBM Business Process Manager workflows
- An IBM Business Process Manager developer can use rich integration features such as Microsoft .NET, Integration Bus Healthcare Connectivity pack, and DFDL
- No changes are required to the existing IBM Business Process Manager programming model

© Copyright IBM Corporation 2016

Figure 19-16. Synergy with Business Process Manager

WM676/ZM6761.0

### Notes:

## Flexible development

- Start with a business process definition
- Process Center snapshots provide an integration handover
  - Snapshot can include multiple service definitions
  - Captured as a .twx file
- WebSphere Integration Developer imports snapshots from the Business Process Manager
  - Provides an implementation of selected definitions
  - Built-in integration tools simplify this activity
- Process Designer reimports updated snapshots from the Integration Bus
  - Completes the business process definition
  - Calls an integration service in the IBM Business Process Manager system activity

© Copyright IBM Corporation 2016

Figure 19-17. Flexible development

WM676/ZM6761.0

### Notes:

IBM Integration Bus supports bottom-up and top-down development. For example, you would complete the following high-level steps to create a Business Process Manager integration service that interoperates with an IBM Integration Bus integration service.

1. Create the Business Process Manager integration service definition.
2. Create a Business Process Manager toolkit, add the integration service definition, and export the toolkit from IBM BPM as an export .twx file.
3. Use the export .twx file to create an IBM Integration Bus integration service.
4. Complete the integration of the IBM Integration Bus integration service in Business Process Manager.



## WebSphere Service Registry and Repository

- Registry holds service metadata
  - Dynamically select services
  - Typically used during maintenance and updates
  - Currently, only web-services-based endpoints
- Repository stores documents (artifacts)
  - WSDL
  - XSD
  - Other XML documents
  - SCDL
- Use the Integration Bus nodes (RegistryLookup and EndpointLookup) to create message flows that retrieve data dynamically from repository
- See “Integrating IBM Integration Bus with WebSphere Service Registry and Repository” white papers in the IBM Integration DeveloperWorks Technical Library:

[www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html](http://www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html)

© Copyright IBM Corporation 2016

Figure 19-18. WebSphere Service Registry and Repository

WM676/ZM6761.0

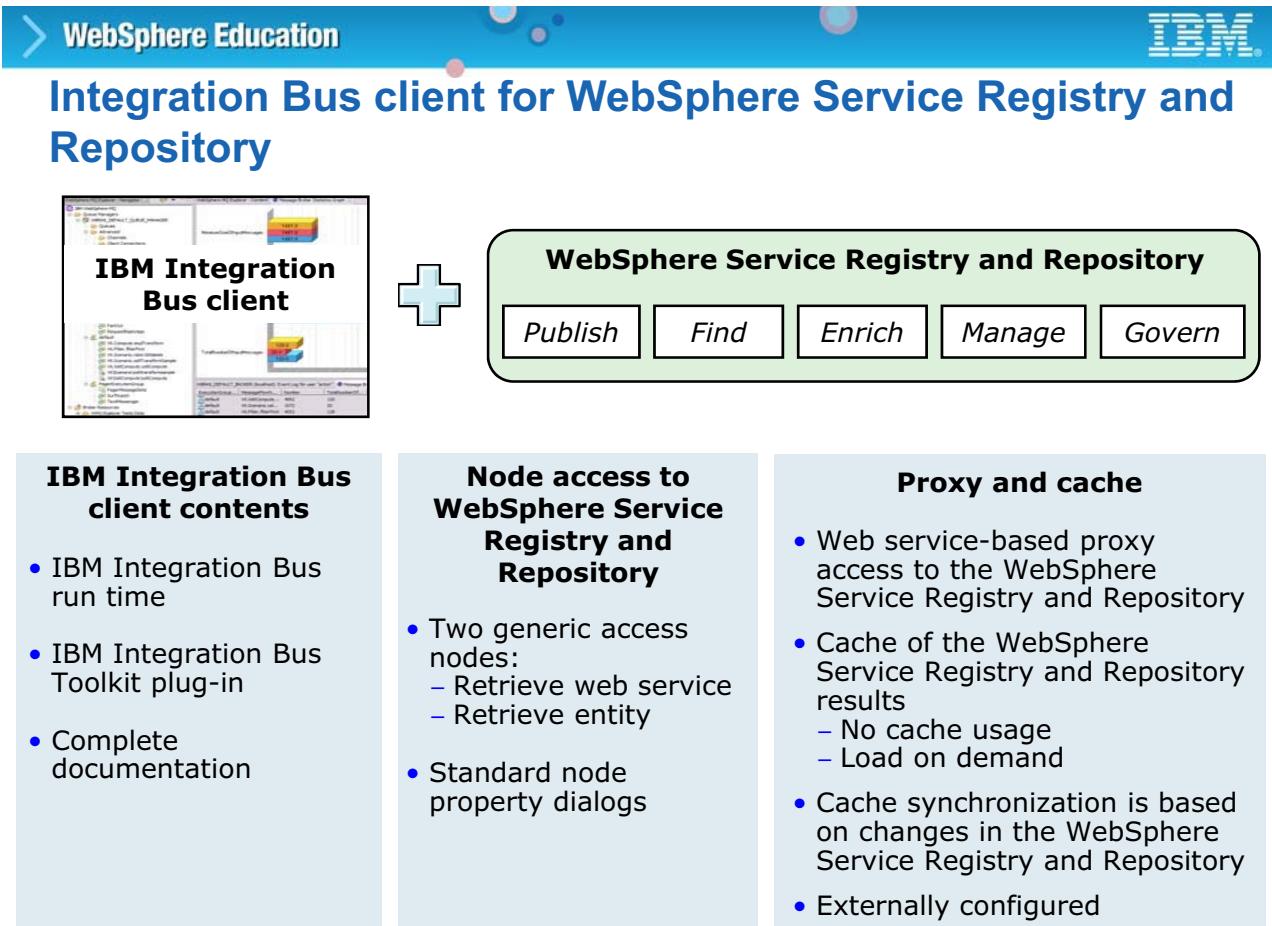
### Notes:

SOA offers the promise of business agility and resilience through reuse, loose coupling, flexibility, interoperability, integration, and governance. These promises are realized by separating service descriptions from their implementations, and by using this descriptive metadata across the service lifecycle. WebSphere Service Registry and Repository provides registry functions that support publication of metadata about the capabilities, requirements, and semantics of services that enable service consumers to find services or to analyze their relationships. It also provides repository functions to store, manage, and assign a version number to service metadata. WebSphere Service Registry and Repository plays a key role in the end-to-end governance underpinnings of the SOA lifecycle.

WebSphere Service Registry and Repository is a central repository of documents that describe services, service interfaces (for example, SOAP over HTTP), and associated policies that control access mechanisms. WebSphere Service Registry and Repository supports the governance of service metadata throughout the lifecycle of a service, from its initial publication in a development space through deployment to service management.

The integration between IBM Integration Bus and WebSphere Service Registry and Repository has both development and runtime aspects. They can be used together to achieve true governance by using WebSphere Service Registry and Repository to determine IBM Integration Bus processing.

Generic XML documents, WSDL, SCDL, and other formats are all storable in the WebSphere Service Registry and Repository, although some queries might apply only to certain document types. For example, a query for a port type can be done only on WSDL documents.



© Copyright IBM Corporation 2016

Figure 19-19. Integration Bus client for WebSphere Service Registry and Repository

WM676/ZM6761.0

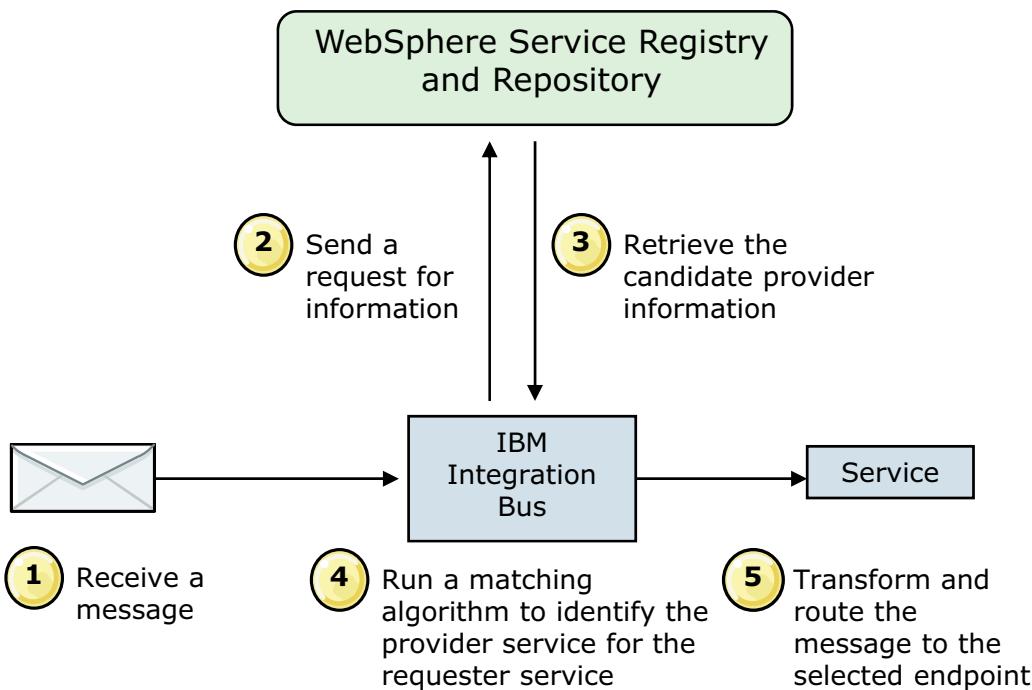
## Notes:

The WebSphere Service Registry and Repository Client nodes are part of IBM Integration Bus; however, the WebSphere Service Registry and Repository server is a separate product.

In the development environment, the WebSphere Service Registry and Repository plug-in can be used to search the registry for a particular entity. This discovered entity can then be used to start the creation of the message set and message flow. You can then use the WSDL drag-and-drop capability in the Toolkit to use the discovered service definition in your flow.

Using the expanded expression support in IBM Integration Bus, the contents of the message in your flow can be used as metadata to dynamically select and start services.

## Dynamic runtime selection and invocation



© Copyright IBM Corporation 2016

Figure 19-20. Dynamic runtime selection and invocation

WM676/ZM6761.0

### Notes:

WebSphere Service Registry and Repository has several basic usage patterns:

- Dynamic XSLT transformation can use the RegistryLookup node to retrieve an XSL stylesheet. A transformation node copies the returned stylesheet into the input message.
- Use the EndpointLookup node to retrieve a set of endpoints between multiple services, for example, choosing between premium service and standard service. A transformation node then selects the required service and copies the endpoint information into the correct place for the SOAP node.
- Have an alternative service provider where the **Failure** terminal is used to denote a problem of accessing services with the SOAP Request node. A transformation node marks the endpoint as failed, returns to back up the flow, and selects the new service.

The figure shows an example of dynamic endpoint selection. An ESB mediation (message flow) is started. This mediation queries WebSphere Service Registry and Repository for information about the requester and candidate provider. The mediation matches the requester with best candidate provider and routes the message.



## IBM DataPower SOA appliances

- IBM DataPower Gateway is a purpose-built security and integration platform for mobile, cloud, API, web, SOA, and business-to-business (B2B) workloads
- Rapidly expand the scope of valuable IT assets to new channels and use cases and reach customers, partners, and employees
- Quickly secure, integrate, control, and optimize access to a range of workloads through a single, extensible, DMZ-ready gateway

© Copyright IBM Corporation 2016

Figure 19-21. IBM DataPower SOA appliances

WM676/ZM6761.0

### Notes:

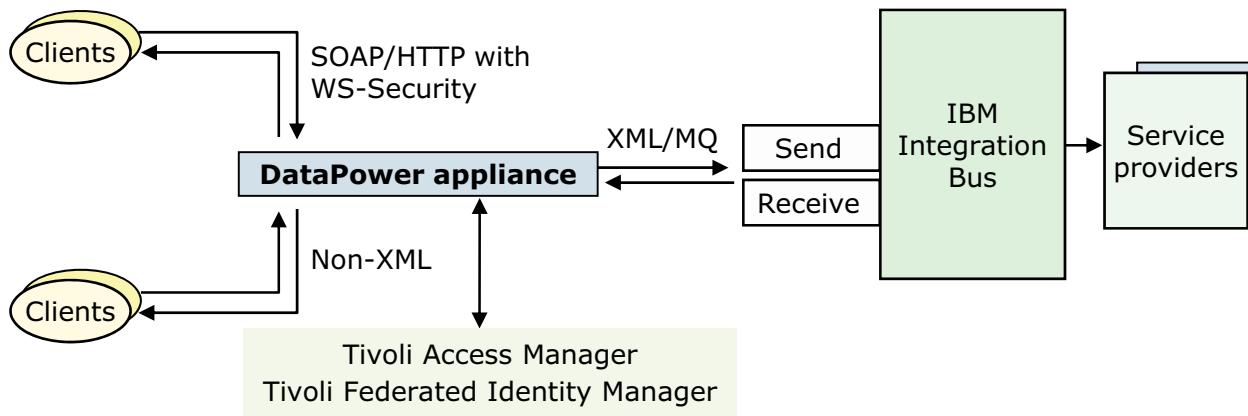
IBM DataPower embeds message processing directly into network hardware. This speed allows complex heavy-use problems to be broken down and approached differently. The devices run as a proxy, providing integration features without core processor impact or application modification. Hardened security is embedded in the devices.

DataPower appliances enable high-volume XML traffic to be processed more effectively, while addressing XML security and integration requirements. DataPower appliances accelerate the deployment of SOA by addressing three potential inhibitors to SOA:

- The complexity of deploying XML and web services with standards that evolve to many different applications and systems
- The additional processing and workload that is associated with a large throughput of XML headers and tags
- The additional security that is needed to protect XML traffic from various threats without burdening existing infrastructures with new, rapidly changing technical requirements.

## DataPower with IBM Integration Bus

- Enhanced WS-\*, and WS-Security support
- Web Services Gateway functions
- Protocol transformations
- High performance non-XML and XML transformations
- Persistent messaging server
- Many standard adapters
- Ability to add Java components or other user-supplied code
- Complex event processing



© Copyright IBM Corporation 2016

Figure 19-22. DataPower with IBM Integration Bus

WM676/ZM6761.0

### Notes:

This scenario shows DataPower working with IBM Integration Bus to provide a high performance web services security gateway.

Clients connect to the DataPower appliance by using SOAP over HTTP. The DataPower appliance connects to IBM Integration Bus by using IBM MQ.

In this scenario, the DataPower device offers significant performance benefits, and allows security processing to be offloaded from the primary application processing within the IBM Integration Bus environment.

The main usages for SOA appliances today are as follows:

- Securely enabling access to the back-end system of record for business partners and customers
- Connecting mainframes or applications to web services or SOA
- Efficiently transforming, routing, and logging messages among applications and web services



## WebSphere Transformation Extender

- Transform, validate, and enrich any document, message or complex data
- In-process data validation delivers trustworthy information for critical business initiatives
- Meet regulatory compliance requirements with predefined data models for industry standards
- Codeless development: Universal reuse and deployment
  - One engine, multiple deployment options
  - Self-describing data model for all data types
- Plug-in node in Integration Toolkit allows for a direct transformation to simplify flows

© Copyright IBM Corporation 2016

Figure 19-23. WebSphere Transformation Extender

WM676/ZM6761.0

### Notes:

WebSphere Transformation Extender is a transformation engine for application data. It has a similar function to the Compute node in IBM Integration Bus, with the advantage of providing a codeless approach to transforming data. WebSphere Transformation Extender is a data mapping and transformation engine. It is unique because it can easily define and map all data types, including complex, deeply nested data types such as EDI, HIPAA, and SWIFT.

Furthermore, WebSphere Transformation Extender can handle transformations in a code-free environment, which can significantly help reduce the time that is required to develop transformations. The WebSphere Transformation Extender embeddable engine can work in various runtimes, one of which is IBM Integration Bus.

By using the WebSphere Transformation Extender design tools, a business analyst or designer can build integration objects across the applications, databases, and systems that are being integrated. All of the extensive integration functions are managed from an easy-to-use, robust, Microsoft Windows design interface.



## WebSphere Transformation Extender Industry Packs

- Enable developers to integrate a range of industry-standard data formats into an existing enterprise infrastructure
- Extends enterprise service bus functions, including the IBM Integration Bus, for industry solutions
- Easily manage and adopt to changing industry standards
  - Healthcare: HIPAA, HL7, and NCPDP
  - Financial services: FIX, NACHA, SEPA, and SWIFTNet
  - EDI: X12, EDIFACT, Odette, and TRADACOMS
  - Insurance: ACORD

© Copyright IBM Corporation 2016

Figure 19-24. WebSphere Transformation Extender Industry Packs

WM676/ZM6761.0

### Notes:

The WebSphere Transformation Extender Industry Packs are an add-on option of predefined data definitions and examples. They are specifically for handling complex transformation that is based on industry standards. You can concentrate on the data application, not changes to a standard.



## Sterling Commerce Connect:Direct

- IBM Sterling Connect:Direct is a managed file transfer product that transfers files between, and within, enterprises
  - Use the Integration Bus CD Input node to receive messages that are transferred to a given Connect:Direct server
  - Connect:Direct server manager notifies the CD Input node when a transfer occurs
- Connect:Direct server manager is defined and runs by using the CDServer configurable service
- Information about file transfers is held on IBM MQ queues
  - Requires a local IBM MQ queue manager that is specified on the integration node

© Copyright IBM Corporation 2016

Figure 19-25. Sterling Commerce Connect:Direct

WM676/ZM6761.0

### Notes:

IBM Sterling Connect:Direct is a managed file transfer product that transfers files between, and within, enterprises.

The CDInput node receives messages that were transferred to a given Connect:Direct server. The node receives both the contents of the file and metadata that is provided by IBM Sterling Connect:Direct on the transfer. One or more CDInput nodes can be used to receive transfers, either in the same flow, different flows, or different integration servers; for any transfer only one CDInput node receives a message.

You can also specify which transfer a CDInput node can receive, by using filters based on the directory and file name of the transfer. After the transfer is processed, you have a set of options for what to do with the transferred file.

The CDInput and CDOOutput nodes go through the Connect:Direct server manager to both send and receive transfers from a server, the name of which is set on the node. The Connect:Direct server manager properties are defined by using a CDServer configurable service.

## Access to mobile services

- IBM MobileFirst Platform Foundation integration makes developing mobile services simple
  - Integration Bus patterns make mobile services integration quick and easy
  - Integration Toolkit supports the conversion of existing Integration Bus services into mobile services
- Windows .NET Mobile Service pattern
  - Integrates mobile applications with IBM Integration Bus .NET web services
  - Simple to configure: Drag the .NET assembly and enter the IBM Worklight adapter details
  - The pattern creates a mobile application to test the IBM Worklight adapter
- Mobile service enablement in the IBM Integration Toolkit
  - Right-click option to mobile-enable the existing Integration Bus services

© Copyright IBM Corporation 2016

Figure 19-26. Access to mobile services

WM676/ZM6761.0

### Notes:

Worklight, an IBM company, provides an advanced mobile application platform and tools software for smartphones and tablets.

Use the Worklight mobile service pattern to integrate a mobile application that is written for Worklight with a service that is running in IBM Integration Bus. You can use the pattern to make an IBM Integration Bus service available through REST APIs used by mobile applications that run on all types of devices.

## Integration Bus Fix Pack enhancements (1 of 2)

- V10.0.0.1
  - Increased support for JDBC drivers
  - WebSphere Enterprise Service Bus conversion tool enhancements
- V10.0.0.2
  - Support for global transactions with CICS
  - Capability to push REST API to an IBM API Management server
  - Use cache transforms in a Mapping node to interact with data that is stored in a global cache
  - Integration Bus can communicate with WebSphere eXtreme Scale grids that use the IBM eXtremelO (XIO) transport mechanism
  - Service trace `temp` option for integration server service trace that automatically switches trace off when the component restarts
  - WebSphere Enterprise Service Bus conversion tool enhancements

© Copyright IBM Corporation 2016

Figure 19-27. Integration Bus Fix Pack enhancements (1 of 2)

WM676/ZM6761.0

### Notes:

This slide and the next summarize the product enhancements for IBM Integration Bus V10 as of the creation of this course. For the most recent information on IBM Integration Bus V10 Fix Packs, see the IBM Knowledge Center.



## Integration Bus Fix Pack enhancements (2 of 2)

- V10.0.0.3
  - Support for Oracle stored procedures
  - Create business transaction monitoring definitions in the IBM Integration web user interface to monitor business transactions across multiple message flows
  - Support for Linux on POWER Little Endian

© Copyright IBM Corporation 2016

Figure 19-28. Integration Bus Fix Pack enhancements (2 of 2)

WM676/ZM6761.0

### Notes:

## Unit summary

Having completed this unit, you should be able to:

- Describe how IBM Integration Bus integrates with other IBM products such as IBM WebSphere Enterprise Service Bus and IBM DataPower Appliances
- Describe how IBM Integration Bus can interact with enterprise information systems

© Copyright IBM Corporation 2016

Figure 19-29. Unit summary

WM676/ZM6761.0

### Notes:



## Checkpoint questions

1. True or False: When developing message flows for use in integration with IBM Business Process Manager, always build the framework in WebSphere Integration Developer first.
  
2. True or False: Configurable services can be modified in the IBM Integration web interface and by using a command.

© Copyright IBM Corporation 2016

Figure 19-30. Checkpoint questions

WM676/ZM6761.0

### Notes:

Write your answers here:

- 1.
  
- 2.



## Checkpoint answers

1. True or False: When developing message flows for use integration with IBM Business Process Manager, always build the framework in WebSphere Integration Developer first.

Answer: **False. Integration Bus supports several various design options.**

2. True or False: Configurable services can be modified in the IBM Integration web interface and by using a command.

Answer: **True.**

© Copyright IBM Corporation 2016

Figure 19-31. Checkpoint answers

WM676/ZM6761.0

### Notes:



# Unit 20. Course summary

## What this unit is about

This unit summarizes the course and provides information for future study.

## What you should be able to do

After completing this unit, you should be able to:

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

## Unit objectives

After completing this unit, you should be able to:

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

© Copyright IBM Corporation 2016

---

Figure 20-1. Unit objectives

WM676/ZM6761.0

### Notes:

## Course learning objectives

After completing this course, you should be able to:

- Use event-driven message processing to control the flow of messages by using message aggregation, message collections, message sequences, and time-sensitive nodes
- Transform data by using Microsoft .NET and XML stylesheets
- Analyze and filter information in complex XML documents
- Extend DFDL message models
- Use message sets and the Message Repository Manager (MRM) parser
- Provide a message flow application as a web service
- Request a web service from within a message flow

© Copyright IBM Corporation 2016

Figure 20-2. Course learning objectives

WM676/ZM6761.0

### Notes:

## Course learning objectives

After completing this course, you should be able to:

- Describe how to implement WS-Addressing and WS-Security standards in IBM Integration Bus
- Create an integration service
- Create and implement an IBM MQ request and response service definition
- Create and implement a database service definition
- Configure security-enabled message processing nodes
- Create a decision service that implements business rules to provide routing, validation, and transformation
- Expose a set of integrations as a RESTful web service

© Copyright IBM Corporation 2016

Figure 20-3. Course learning objectives

WM676/ZM6761.0

### Notes:

## Course learning objectives

After completing this course, you should be able to:

- Use a global cache to store static data
- Record and replay data that a message flow application processes
- Implement publish and subscribe with IBM Integration Bus
- Describe the workload management options for adjusting the message processing speed, and controlling the actions that are taken on unresponsive flows and threads
- Construct user-defined patterns
- Describe how IBM Integration Bus integrates with other IBM products such as IBM WebSphere Enterprise Service Bus and IBM DataPower Appliances

© Copyright IBM Corporation 2016

Figure 20-4. Course learning objectives

WM676/ZM6761.0

### Notes:



## To learn more on the subject

- IBM Training website:

[www.ibm.com/training](http://www.ibm.com/training)

- Learn about IBM Integration Bus:

[www.ibm.com/developerworks/community/blogs/c7e1448b-9651-456c-9924-f78bec90d2c2/entry/learn\\_about\\_ibm\\_integration\\_bus?lang=en](http://www.ibm.com/developerworks/community/blogs/c7e1448b-9651-456c-9924-f78bec90d2c2/entry/learn_about_ibm_integration_bus?lang=en)

- IBM Integration Bus community on DeveloperWorks:

[www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html](http://www.ibm.com/developerworks/websphere/zones/businessintegration/wmb.html)

- IBM Knowledge Center for IBM Integration Bus:

[www.ibm.com/support/knowledgecenter/SSMKHH\\_10.0.0/com.ibm.etools.msgbroker.helphome.doc/help\\_home\\_msgbroker.htm](http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.msgbroker.helphome.doc/help_home_msgbroker.htm)

© Copyright IBM Corporation 2016

Figure 20-5. To learn more on the subject

WM676/ZM6761.0

## Notes:



## Unit summary

Having completed this unit, you should be able to:

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

© Copyright IBM Corporation 2016

Figure 20-6. Unit summary

WM676/ZM6761.0

### Notes:



# Appendix A. List of abbreviations

<b>ACORD</b>	Association for Cooperative Operations Research and Development
<b>AIX</b>	Advanced IBM UNIX
<b>API</b>	application programming interface
<b>ASP</b>	auxiliary storage pool
<b>ASCII</b>	American Standard Code for Information Interchange
<b>ATM</b>	automated teller machine
<b>B2B</b>	business-to-business
<b>BAPI</b>	Business Application Programming Interface
<b>BAR</b>	broker archive
<b>BCD</b>	binary-coded decimal
<b>BLOB</b>	binary large object
<b>CCD</b>	client channel definition
<b>CCD</b>	consistent-change-data
<b>CCDT</b>	client channel definition table
<b>CDA</b>	Clinical Document Architecture
<b>CEI</b>	Common Event Infrastructure
<b>CICS</b>	Customer Information Control System
<b>CIL</b>	Common Intermediate Language
<b>CLI</b>	command-line interface
<b>CLI</b>	common language infrastructure
<b>CLR</b>	common language runtime
<b>CLS</b>	Common Language Specification
<b>COBOL</b>	Common Business Oriented Language
<b>COM</b>	Component Object Model
<b>CORS</b>	Cross-Origin Resource Sharing
<b>CSV</b>	comma-separated values
<b>CTS</b>	Common Type System
<b>CWF</b>	Custom Wire Format
<b>DFDL</b>	Data Format Description Language
<b>DICOM</b>	Digital Imaging and Communication in Medicine
<b>DLR</b>	Dynamic Language Runtime

<b>DMZ</b>	demilitarized zone
<b>DSN</b>	data source name
<b>DTD</b>	document type definition
<b>EDI</b>	electronic data interchange
<b>EDIFACT</b>	Electronic Data Interchange For Administration, Commerce and Transport
<b>EIS</b>	enterprise information system
<b>EPR</b>	endpoint reference
<b>ERP</b>	engineering resource planning
<b>ESB</b>	enterprise service bus
<b>ESQL</b>	Extended Structured Query Language
<b>FIX</b>	Financial Information Exchange
<b>GB</b>	gigabyte
<b>HIPAA</b>	Health Insurance Portability and Accountability Act
<b>HL7</b>	Health Level 7
<b>HR</b>	human resources
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>IBM</b>	International Business Machines Corporation
<b>IIC</b>	Institution Identification Code
<b>IP</b>	Internet Protocol
<b>ISO</b>	International Organization for Standardization
<b>IT</b>	information technology
<b>JAR</b>	Java archive
<b>JCA</b>	Java EE Connector Architecture
<b>JDBC</b>	Java Database Connectivity
<b>JIT</b>	just-in-time
<b>JMS</b>	Java Message Service
<b>JSON</b>	JavaScript Object Notation
<b>JVM</b>	Java virtual machine
<b>KDC</b>	Kerberos Key Distribution Center
<b>KPI</b>	key performance indicator
<b>LDAP</b>	Lightweight Directory Access Protocol
<b>LINQ</b>	Language Integrated Query

---

<b>MAP</b>	Message Addressing Property
<b>MQ</b>	Message Queue
<b>MQMD</b>	IBM MQ Message Descriptor
<b>MQTT</b>	MQ Telemetry Transport
<b>MRM</b>	Message Repository Manager
<b>MSIL</b>	Microsoft intermediate language
<b>MTOM</b>	Message Transmission Optimization Mechanism
<b>NCPDP</b>	National Council for Prescription Drug Programs
<b>NGEN</b>	Next Generation
<b>ODBC</b>	Open Database Connectivity
<b>ORU</b>	observation result
<b>OSLC</b>	Open Services for Lifecycle Collaboration
<b>PEP</b>	policy endpoint
<b>POJO</b>	plain old Java object
<b>QA</b>	quality assurance
<b>QOS</b>	quality of service
<b>REST</b>	Representational State Transfer
<b>RFC</b>	request for change
<b>RPC</b>	Remote Procedure Call
<b>SAML</b>	Security Assertion Markup Language
<b>SCA</b>	Service Component Architecture
<b>SCDL</b>	Service Component Description Language
<b>SCI</b>	Scalable Coherent Interconnect
<b>SCI</b>	Structured Call Interface
<b>SEPA</b>	Software Engineering Process Authority
<b>SOA</b>	service-oriented architecture
<b>SQL</b>	Structured Query Language
<b>SSL</b>	Secure Sockets Layer
<b>STS</b>	Security Token Service
<b>SwA</b>	SOAP with Attachments
<b>SWIFT</b>	Society for Worldwide Interbank Financial Telecommunication
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>TDS</b>	Tagged/Delimited String (Format)

<b>TLOG</b>	transaction log
<b>TRADACOMS</b>	Trading Data Communications
<b>UDDI</b>	Universal Description, Discovery, and Integration
<b>UNIX</b>	Uniplexed Information and Computing System
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>UUID</b>	Universally Unique Identifier
<b>VB</b>	Visual Basic
<b>W3C</b>	World Wide Web Consortium
<b>WAR</b>	web archive
<b>WCF</b>	Windows Communication Foundation
<b>WLM</b>	Workload Manager
<b>WPF</b>	Windows Presentation Foundation
<b>WS</b>	web service
<b>WSDL</b>	Web Services Description Language
<b>XDF</b>	eXtreme data format
<b>XIO</b>	eXtremelO
<b>XML</b>	Extensible Markup Language
<b>XOM</b>	XML Object Model
<b>XSD</b>	XML Schema Definition
<b>XSL</b>	Extensible Stylesheet Language
<b>XSLT</b>	Extensible Stylesheet Language Transformations



**IBM**  
®